

**Personalized Autism Infographics: A Web Development  
Project with and for Autistic People**

**Bachelorarbeit**

der Philosophisch-naturwissenschaftlichen Fakultät  
der Universität Bern

vorgelegt von

Sara Peeters

2018

Leiter der Arbeit:  
Prof. Dr. Oscar Nierstrasz  
Institut für Informatik

## Abstract

Autism is a lifelong neurodevelopmental condition influencing a person's social interaction, communication and sensory perception. Autistic people tend to need a certain level of structure and routine and can often be overwhelmed in day-to-day situations. The fact that autistic people experience the world differently and have different needs than the non-autistic majority often leads to misunderstandings and even conflict situations.

So called autism id cards aim to help in such situations, where communication between those unfamiliar with autism and autistic people fails, as well as to raise awareness in general. Their usefulness is, however, limited by the fact that the autism spectrum is rather broad, and the explanations on the cards are very general.

This project wants to build on the idea of autism id cards by developing a digitalized, personalizable version of them. It consists of a web app where autistic people can click and write together one or more personalized infographics and make them available to the people they choose through an automatically generated link, or save them on their phone.

The project is iteratively developed in partnership with the psychiatric department of the hospital of Frutigen, Meiringen, Interlaken. The input of autistic adults, parents of autistic children, friends and caretakers as potential users is taken into account at various stages of the development through surveys and usability testing.

Through the modularity of the infographics and the free text options in each module, the possibilities for personalization are endless. At the same time the preformulated statements, and the example infographics that can be used as a starting point for customization, provide enough guidance for users not to get lost.

While the end result is not yet ready for large scale public use, it is a solid proof of concept with extensive functionality.

## Preface

After having studied Photonics engineering and trying my hand at experimental physics research, I rediscovered Computer Science a couple of years ago. At about the same time, I was diagnosed autistic. Since then, not only software development, but also autism, neurodiversity and the autistic community have become major interests of mine. I am, therefore, grateful to have gotten the opportunity to combine them in this Bachelor project.

This thesis is one of the outcomes of the project. It can be read on its own, but for a complete view of the work done, it should be combined with the project's source code (<https://github.com/aspigirlcodes/uniqid>) and the final working version of the product (<https://project.aspigirlcodes.ch/>). The source code of the tutorial in Appendix A, which was written to fulfill the requirement of the course "Anleitung zu wissenschaftlichen Arbeiten", can be found in its own github repository: <https://github.com/aspigirlcodes/django-serviceworker-tutorial>.

Throughout my involvement in the autistic community I have experienced more than ever how important the use of inclusive language is. For this reason, I have chosen to use identity first language when autistic people are concerned. This coincides with the preferences of autistic adults as well as those of a considerable proportion of other participants of the autism community [32, 60].

For similar reasons, I will be using the gender inclusive singular "they" when referring to individual persons (mainly users) of unknown or unspecified gender, even though this practice is not yet generally accepted in formal writing [23].

I want to thank Prof. Nierstrasz and the University of Bern for giving me the opportunity to successfully finish my bachelors studies by providing me with the accommodations I needed as an autistic student, as well as Ms. Schwarz and Dr. med. Niemeyer for being the project's customers and partners. I also want to thank the participants of questionnaires and user testing for their time. Special thanks go to those people who supported me with communication and other tasks I could not manage on my own.

Sara Peeters  
Bern, March 26, 2018

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| 1.1      | Introduction to Autism . . . . .                                      | 1         |
| 1.1.1    | Models of Disability . . . . .  | 1         |
| 1.1.2    | Defining Autism . . . . .   | 2         |
| 1.1.3    | Neurodiversity . . . . .  | 3         |
| 1.2      | Project Description . . . . .   | 4         |
| 1.2.1    | Problem . . . . .   | 4         |
| 1.2.2    | Solution . . . . .  | 5         |
| 1.2.3    | Use Cases . . . . .   | 5         |
| 1.2.4    | Partners and Stakeholders . . . . .                                   | 6         |
| 1.2.5    | Thesis Overview . . . . .   | 6         |
| <b>2</b> | <b>Current Knowledge on Web Design for and with Autistic People</b>   | <b>7</b>  |
| 2.1      | Internet Usage by Autistic People . . . . .                           | 7         |
| 2.2      | Accessibility for Autistic People . . . . .                           | 8         |
| 2.3      | User Centered and Participatory Design with Autistic People . . . . . | 8         |
| <b>3</b> | <b>Towards a First Prototype</b>                                      | <b>10</b> |
| 3.1      | Preliminary User Research . . . . .                                   | 10        |
| 3.2      | Developing Relevant Modules . . . . .                                 | 11        |
| 3.2.1    | Existing Research . . . . .   | 11        |
| 3.2.2    | Module Creation . . . . .   | 12        |
| 3.2.3    | Paper Prototypes of Modules . . . . .                                 | 14        |
| 3.3      | Creating Example Infographics . . . . .                               | 14        |
| <b>4</b> | <b>Technical Implementation and Challenges</b>                        | <b>15</b> |
| 4.1      | Introduction . . . . .  | 15        |
| 4.2      | Django . . . . .  | 15        |
| 4.3      | Objects versus Database Tables . . . . .                              | 16        |
| 4.3.1    | Problem Description . . . . .   | 16        |
| 4.3.2    | Solutions . . . . .   | 17        |
| 4.3.3    | Proposed Model Structure . . . . .                                    | 18        |
| 4.4      | User Authentication . . . . .   | 20        |
| 4.4.1    | User Authentication in Django . . . . .                               | 20        |
| 4.4.2    | Registration and Login Design for this Project . . . . .              | 20        |

---

|          |  |           |
|----------|--|-----------|
| 4.4.3    | Password Reset Links in Django . . . . .                                     | 22        |
| 4.4.4    | Using Django's Password Reset Mechanism for Different Purposes . . . . .     | 24        |
| <b>5</b> | <b>Validation</b>  | <b>26</b> |
| 5.1      | Version 0.1 . . . . .  | 26        |
| 5.2      | Version 0.2 and 0.3 . . . . .  | 27        |
| 5.3      | Version 0.4 . . . . .  | 27        |
| 5.3.1    | Call to Action Button in the Navigation Bar . . . . .                        | 27        |
| 5.3.2    | The purpose of the Web Application . . . . .                                 | 28        |
| 5.3.3    | Previewing the Work During Creation . . . . .                                | 28        |
| 5.3.4    | Scrolling Needed to Find a button . . . . .                                  | 31        |
| 5.3.5    | Registration Process . . . . .   | 31        |
| 5.3.6    | Cloning Examples as a Starting Point . . . . .                               | 31        |
| 5.4      | Version 0.4.1 . . . . .  | 31        |
| 5.5      | Version 0.5 . . . . .  | 32        |
| <b>6</b> | <b>Conclusion</b>  | <b>33</b> |
| 6.1      | Discussion . . . . .   | 33        |
| 6.2      | Future work . . . . .  | 34        |
| <b>A</b> | <b>Tutorial: Adding Service Workers to your Django App</b>                   | <b>36</b> |
| A.1      | Introduction . . . . .   | 36        |
| A.2      | Prerequisites . . . . .  | 37        |
| A.3      | Preparation: Improve Django Templates to Output a Proper HTML Page . . . . . | 39        |
| A.4      | Adding a Service Worker to Return a Simple Offline Message. . . . .          | 41        |
| A.4.1    | Registering Your First Service Worker . . . . .                              | 41        |
| A.4.2    | Letting Your Service Worker Serve an Offline HTML Page . . . . .             | 43        |
| A.4.3    | Intermezzo: Service Worker Life Cycle . . . . .                              | 45        |
| A.4.4    | Caching Multiple Assets and Cleaning up Old Caches . . . . .                 | 47        |
| A.5      | Implementing a Caching Strategy . . . . .                                    | 51        |
| A.5.1    | Caching Patterns . . . . .   | 51        |
| A.5.2    | Caching the Polls App . . . . .  | 51        |
| A.5.3    | Letting the User Know the Content Might be Outdated . . . . .                | 53        |
| A.6      | What is Next? . . . . .  | 57        |
| <b>B</b> | <b>Static Example Infographics Using HTML and Bootstrap</b>                  | <b>58</b> |
|          | <b>List of Tables</b>  | <b>61</b> |
|          | <b>List of Figures</b>   | <b>61</b> |
|          | <b>Bibliography</b>  | <b>62</b> |

# Chapter 1

## Introduction

This chapter aims to introduce both the context of this project, autism, and the project itself.

### 1.1 Introduction to Autism

While not every autistic person experiences their autism as a disability, it is often classified as one in different contexts including legally, as well as by professionals and autistic people themselves [4]. It therefore makes sense to first have a look at definitions and understandings of disability, before moving on to try and define autism. After a short explanation of different models of disability, I will present the medical definition of autism used for diagnosis. Because this definition has clear shortcomings, I will also introduce a broader perspective by illustrating alternative definitions and introducing the terminology around neurodiversity.

#### 1.1.1 Models of Disability

Historically, disability has mostly been defined and measured by the physical or cognitive impairments disabled people experience. This way of thinking is known as the medical model of disability [29, 46], and it still has a big impact on the way we deal with it. In its attempts to improve the lives of people with disabilities, the medical model focuses on attacking the impairments. It proposes cures and medical interventions, (almost) without considering alternatives or costs.

The social model of disability [29, 46] on the other hand, states that social injustice and discrimination of disabled people, rather than the impairments or differences themselves, are the cause of disability. Therefore, this model demands social change in order to improve the life quality of disabled people. From this point of view, acceptance and adaptation by the society removes barriers and mitigates the impact of disability to a large extend.

Many organizations and entities now use a mixture of both models to approach disability. The World Health Organization (WHO) for example states that “[disability] is a complex phenomenon, reflecting the interaction between features of a persons body and features of the society in which he or she lives. Overcoming the difficulties faced by people with disabilities requires interventions to remove environmental and social barriers.” [44]

## 1.1.2 Defining Autism

### Medical Definitions as a Basis for Diagnosis

Autism is listed in two diagnostic manuals: the Diagnostic and Statistical Manual of Mental Disorders, Fifth Edition (DSM-5) [1] and the International Statistical Classification of Diseases and Related Health Problems, Tenth Revision (ICD-10) [45] with very similar definitions; They both classify autism as a neurodevelopmental disorder that “[...] is characterized by persistent deficits in social communication and interaction across multiple contexts, as well as restricted, repetitive patterns of behavior, interests, or activities.” [1]

One of the most important properties of both manuals’ definitions is that they are entirely based on observed behavior. The behavior considered not normal is then divided into different categories such as repetitive behavior, special interests, communication and social interaction difficulties.

Throughout the years, different sub-groups of autism related diagnoses have emerged and disappeared again based on how well someone’s considered abnormal behavior fits in the defined categories and how strong its abnormality is perceived [27]. Beyond this categorization, there is a tendency towards wanting to rate a person’s functioning ability as a fixed unequivocal quantity during the diagnostic process [57, 66].

While those properties might seem useful to diagnosticians, they cannot be considered a complete definition of autism, and are in some ways even harmful to autistic people. Besides those problems, recent research suggests that the focus on behavior of autistic white boys at the base of those diagnostic criteria causes significant bias [6, 34].

### Other Definitions

More useful autism definitions try to improve on the diagnostic definition in a number of ways. These improvements might include, but are not limited to:

Going beyond just categorizing behavior and trying to understand how the behavioral differences are related to different perception and processing of information by the brain. Those definitions will often include different sensory perception and processing by autistic people, something that is considered a key factor of autistic experience.

In the light of relating the different behavior to different brain processing, they also tend to avoid just talking about deficits. Instead they try to find a logical explanation for each difference taking into account the different neurology.

On top of that, they often recognize that functioning ability varies with time and depends strongly on the situation, and therefore avoid the use of fixed functioning labels.

Here are two examples of such definitions:

“Autistic individuals share a neurological type, which is qualitatively different to that of non-autistics, and which will necessarily impact, both positively and negatively, on aspects of their thinking and learning; sensory processing; social relational experiences; and communicative style, abilities and preferences. An autistic person’s experience

of and ability to be successful in the world, will be dependent on the closeness of compatibility, between their individual profile of skills and requirements and their physical and social environment. Levels of sensitivity to environmental factors vary between individuals, and within the same individual over time, so that the presentation of autism is ever changing. A person's neurological type, however, remains constant, and being autistic is a lifelong identity."

Julia Leatherland, Autism researcher [5]

"Autism is a neurological variation that occurs in about one percent of the population and is classified as a developmental disability. Although it may be more common than previously thought, it is not a new condition and exists in all parts of the world, in both children and adults of all ages. The terms "autistic" and "autism spectrum" often are used to refer inclusively to people who have an official diagnosis on the autism spectrum or who self-identify with the autistic community. While all autistics are as unique as any other human beings, they share some characteristics typical of autism in common: Different sensory experiences. [...] Non-standard ways of learning and approaching problem solving. [...] Atypical, sometimes repetitive, movement. [...] Difficulties in understanding and expressing language as used in typical communication, both verbal and non-verbal. [...] Difficulties in understanding and expressing typical social interaction. [...]"

Autism Self Advocacy Network, About autism [39]

### 1.1.3 Neurodiversity

The topic of neurodiversity is discussed in many scientific as well as popular publications [31, 22, 65, 3]. Below explanations are based on those publications.

The neurodiversity paradigm is strongly related to the social model of disability. It is based on the understanding that autism and other neurological differences such as ADHD, bipolarity and dyspraxia, are the result of natural variations in human neurology, rather than pathological conditions. Based on this premise the neurodiversity movement campaigns for the civil rights of people belonging to those neurological minorities. The neurodiversity terminology was first formulated by Judy Singer in 1998[61], based on ideas influenced by other autistic rights activists such as Jim Sinclair [59] and Martijn Dekker [16].

The gist of the neurodiversity way of thinking might be summarized as follows; Neurodivergent people's brains function differently than those of the neurotypical majority our society is designed for. Therefore, they might have difficulties functioning according to the expectations of this society. But despite these difficulties, they do not need to be cured in the sense of normalization, or prevented in the sense of eugenics. Rather than medical interventions, they might need support to improve their quality of life while also protecting their neurological integrity.

## 1.2 Project Description

### 1.2.1 Problem

While there is no lack of public interest in autism, there does seem to be a lack in knowledge and general understanding of how to interact with autistic people. Instead, the common attitude towards them is based on preconceptions that are either very simplified or plain wrong. But if people's ideas about the nature of the problem are skewed, they will be less inclined to accept different behavior or recognize different needs, let alone be able and willing to provide appropriate support [55]. Therefore, this lack of knowledge can have a significant impact on autistic peoples wellbeing.

From the previous explanations about autism and neurodiversity, it should be clear that misunderstandings and difficult situations do happen, and that it is not desirable to try to avoid them by normalizing autistic people. One could go further and say that not so much the misunderstandings are a problem, but the fact that, being part of a minority group, the burden of communication in difficult situations is always with the autistic person, putting them under unnecessary pressure.



Figure 1.1: Examples of autism id cards

These problems are not new, and people have attempted to solve them before. The most prevalent solution currently used are so called autism id cards (some examples are depicted in Figure 1.1).

They aim to help in situations where communication between those unfamiliar with autism and autistic people fails, as well as to raise awareness in general. They usually list some typical communication difficulties of autistic people, have a place for the name of the cardholder and sometimes for contact data of a person to be contacted in case of emergencies.

The usefulness of those cards is, however, limited by the fact that the autism spectrum is rather broad, and the explanations on the cards very general. Also, as space is limited, there is not much place for personalization. People who do not identify with what is on the card will rarely carry it with them.

### 1.2.2 Solution

This project aims to improve the current solution of autism id cards by developing a digitalized, personalizable version of them. It exists of a web app where autistic people can click and write together one or more personalized infographics and make them available to the people they choose through an automatically generated link, or save them on their phone.

The infographics consist of modules which are mostly optional. Examples of possible modules are: “general information”, “preferred ways of communication and communication difficulties”, “things I’m (not) good at”, “what helps me when I’m stressed”, “medical info”, “contact information”, etc. For each module there are predefined choices as well as free text options.

Through their modularity and high level of personalization as well as the possibility to create different infographics for different occasions, the application area of this product is deliberately kept broad.

### 1.2.3 Use Cases

The solution can be used in a wide range of situations where communication about autism is relevant. These include disclosure of the diagnosis to peers, any situation where autistic people need to or want to ask for accommodations, as well as medical emergencies or situations where the behavior of the autistic person attracts or is expected to attract negative attention. Some specific examples are given below.

Going to a new hairdresser or doctor or taking part in a workshop or course are all situations where it could be beneficial for the professional the autistic person will be working with to know something about autism and how it affects this specific person in this specific situation. Each of those situations might, however, requires different kinds of information or a different level of detail.

A medical emergency is a stressful situation, in which autistic people may behave unexpectedly and have trouble to communicate effectively. If the doctors do not know the patient is autistic, they may interpret their behavior completely wrong, resulting in the patient not receiving proper treatment. Besides the use of emergency chat applications as a text based alternative to speech [38], a detailed description of the autistic person’s differences and needs can inform the doctors without requiring the autistic person to engage in communication at all.

Going on vacation abroad with autistic children might cause the need to communicate about autism in different languages. The ability to change the languages of the infographics would make the product very useful for this use case.

#### **1.2.4 Partners and Stakeholders**

This project is carried out in collaboration with lic. phil. Edith Schwarz and Dr. med. Tim Niemeyer from the psychiatric department of the Hospital of Frutigen Meiringen Interlaken (FMI) in Interlaken. They shared their professional expertise and helped defining the requirements.

Three main groups of stakeholders were identified. First, there are the autistic adults and children, who are at the center of the project. Second, there are their parents, allies or caretakers, who might make infographics on their behalf or assist them in the creation process. And third, there are the people who view and interpret the infographics shared with them by the first two groups.

#### **1.2.5 Thesis Overview**

This chapter introduced both the context and the content of the project described in this thesis. The next chapter will explore the relationship between the autistic population and the internet and web development by means of a literature study. The third chapter of this thesis will focus on the development of the prototype in collaboration with partners and stakeholders. Chapter 4 is reserved for technological content related to the project. Rather than keeping the chronological, protocolary pace of the previous chapter, it will introduce the main technologies used, and then describe two technological challenges, and how they were solved. Chapter 5 is dedicated to the validation of the product throughout the project and the resulting improvements. Chapter 6, finally, contains a discussion of the work done as well as the anticipated next steps.

## Chapter 2

# Current Knowledge on Web Design for and with Autistic People

### 2.1 Internet Usage by Autistic People

From the beginning, the internet has played an important role in the appearance of the first autistic communities and autistic lead organizations. Thanks to the internet, autistic adults could meet others like them on their own terms more easily, through mailing lists, forums and IRC channels, [49, 28, 58, 16].

Soon thereafter, they started taking over part of the autism narrative, that had until then been fixed in the hands of organizations lead by parents and professionals. Autism Network International, the first autistic lead organization worldwide, was founded, and autistic voices started to be heard more and louder [58]. Judy Singer, the autistic activist known for coining the word neurodiversity, stated that “The impact of the internet on autistics may one day be compared to the spread of sign language among the deaf.” [62]

Until today, the internet remains a more comfortable communication medium than face-to-face oral communication for many autistics. Reasons why autistic people may be more comfortable communicating online include the fact that some prefer written communication in general, or that they can contact people with shared interests living further away. Moreover, the absence of confusing nonverbal cues can be a relief, as well as the pace of online communication being different and adaptable to fit ones needs [11, 15, 7].

But the internet, and technology in general, is not only important for its use to stay in touch with peers. It can also lower barriers and help autistic people to live free and independently in the real world. For example, it does provide a lot of information upfront, that before the internet, could only be found out real time or through oral communication. An example of a project exploiting this by providing even more information specifically targeted towards autistic students is the “Autism&Uni project” [19]. This project, funded by the European Commission,

is aimed at designing tools to help autistic people gain access to and succeed in higher education. To do this they built a toolkit with which universities can generate a website with university specific information aimed at autistic students. Topics include telling the university about autism, managing expectations, orientation on campus and typical study situations.

## 2.2 Accessibility for Autistic People

The above section illustrates that the internet is accessible to a part of the autistic population as well as that it has specific benefits for those who can use it. It is, therefore, desirable to improve the accessibility and usability of the web for autistic people, so that more of them can use it with more ease.

A first step towards a more accessible web for autistic people is to implement accessibility recommendations for other disabilities. Not only does being autistic not prevent you from not also having another disability, some disabilities such as dyspraxia and learning disabilities are also more common in the autistic population [43, 17]. On top of that, autistic users might use accessibility helpers meant for other disabilities such as screen readers or subtitles to deal with content that is hard to process, too overwhelming or distracting [33].

If you want to go further and think about autism specific accessibility, you can find extensive guidelines in Pavlov's paper on the topic [47], or in a more condensed form at the website of the National Autistic Society (UK)[63]. When looking through those guidelines one finds out that it all boils down to one fundamental rule: incorporate structure and avoid sensory overload. As an added bonus it is often recommended to allow for some personalization of the layout, to accommodate even more people with different sensory and information processing styles.

## 2.3 User Centered and Participatory Design with Autistic People

User centered design describes a design process where users can take influence. There are several ways users can be involved, and also the time in the design process and the intensity of the involvement can vary. Important is that the needs and interests of the user are recognized and usability is in focus [42]. Participatory design is a specific form of user centered design where the users actively participate in the design team throughout the design process, and cooperate at eye level with the designers [56].

There is a substantial number of papers on involving autistic people in the design process of products where they are the target audience. A lot of them, however, focus on children and autistic people with learning disabilities and severe communication problems [37, 14, 13]. The approaches they propose are beyond the scope this project, where I will assume that those people will interact with the web app with help of a guardian or caretaker.

Interesting frameworks developed for participatory design with autistic children are the IDEAS [10, 8] and Diversity for Design (D4D) framework [9]. Both incorporate elements of the TEACCH methodology, an approach used in education of autistic children. The aim of TEACCH is to

provide a structured environment with specific supports tailored to the individual, allowing them to function as comfortably and effectively as possible.

The frameworks propose structured meetings, where the agenda of planned activities is visible for the participants and items are ticked off as the meeting goes along. They also establish a connection with the participants' hobbies. On top of that, in order to prevent anxiety of very open design activities, they start off with more guided activities, where small changes can be made to existing designs, before putting the participants in front of an empty piece of paper.

Concerning participatory design with autistic adults, interesting findings came out of the Autism&Uni project [20, 19]. They used the 5-step model for design thinking by Stanford University's d.school. This method provides a solution-based human-centered approach suited for solving complex design problems that is not aimed specifically at autistic or disabled people. It divides the design process into 5 steps: Empathise, define, ideate, prototype and test [48]. The researchers of the Autism&Uni project found that the method provided an effective way for involving intellectually able autistic adults in the design process. They suggest that the approach to data collection should be carefully considered when working with autistic adults. Indeed, providing multiple ways of participation (questionnaires, small group workshops, one-to-one sessions, ...) will result in a more diverse group of autistic participants.

It seems that, for this target group, the accessibility of the participation event matters more than the accessibility of the design method. This means that online surveys should follow the accessibility standards mentioned in the previous section. Face-to-face meetings can be made more accessible by allowing users to see the place of the meeting in advance, giving concrete rather than abstract information, and accounting for possible sensory issues. Furthermore, organizers should be prepared to make individual adjustments on request, if possible [63].

## Chapter 3

# Towards a First Prototype

### 3.1 Preliminary User Research

After planning the project, generating ideas based on my own experience as an autistic adult as well as the experience of Ms. Schwarz and Dr. med. Niemeyer as autism professionals, and doing literature research, there was one thing left to be done before the creation process could really start. I had to validate that the needs that I had anticipated based on the knowledge I had gathered, were also present in real prospective users. In order to do this, I created a small survey and distributed it among 10 autistic adults. It covered two topics: their experience with the internet and what kind of websites they liked and disliked in particular, and their experiences with talking about autism in different situations.

Only three of them sent back the filled out survey. Given the time frame and the limited choice of interaction (written survey with open questions only), this was not unexpected. It was therefore good that I could also fall back on user research done by the AASPIRE Healthcare project [53, 40, 41], and the Autism&Uni project [20, 19]. Both projects have done more extensive user research and have a group of target users similar to this project.

All three respondents liked clearly structured websites, not containing too much information, but arranging it really well. They disliked websites that bombarded them with news or other information they could hardly process or blend out. They also did not like it when the information was not well compartmentalized. For example, one person explained she did not like websites where you could go from one topic to the next by scrolling. This technique, that is often applied so that people can get an overview of multiple topics without the overhead of clicking, also smudges the boundaries between topics, which can be confusing to users who rely on those boundaries to process the information.

All respondents did have experience with explaining autism in different situations. However, none of them had good external resources they could rely on to do so. Some carried an autism id card for emergency situations, others did not but still said they would find it useful. No one was completely satisfied with the content of the cards, though, regardless of whether or not they carried one with them. They also talked about a wide range of different things they would want others to know about how they experienced autism and the related problems they had, depending

on the person and the situation. This makes clear that a tool that would possibly help them communicate about autism, would certainly need to be flexible and personalizable.

## 3.2 Developing Relevant Modules

### 3.2.1 Existing Research

The Academic Autistic Spectrum Partnership In Research and Education (AASPIRE) is a partnership that aims to bring together people from the autistic and academic communities in the USA to perform research projects that are relevant to autistic adults [51]. One of the projects they realized between 2011 and 2016 is the AASPIRE Healthcare Toolkit, containing a tool to create a personalized accommodations report that autistic people can give their healthcare provider to explain communication, sensory and other issues and suggest accommodations that could help them [53].

Like all AASPIRE projects, the development of this tool was done through Community Based Participatory Research (CBPR)[52]. Following this methodology, the creators of the tool have put a large amount of effort in researching the problems autistic people experience in the context of healthcare. They have categorized those problems in well defined categories and formulated a large set of descriptions for users to choose from to describe the problems they experience or the accommodations they would like to benefit from [40, 41].

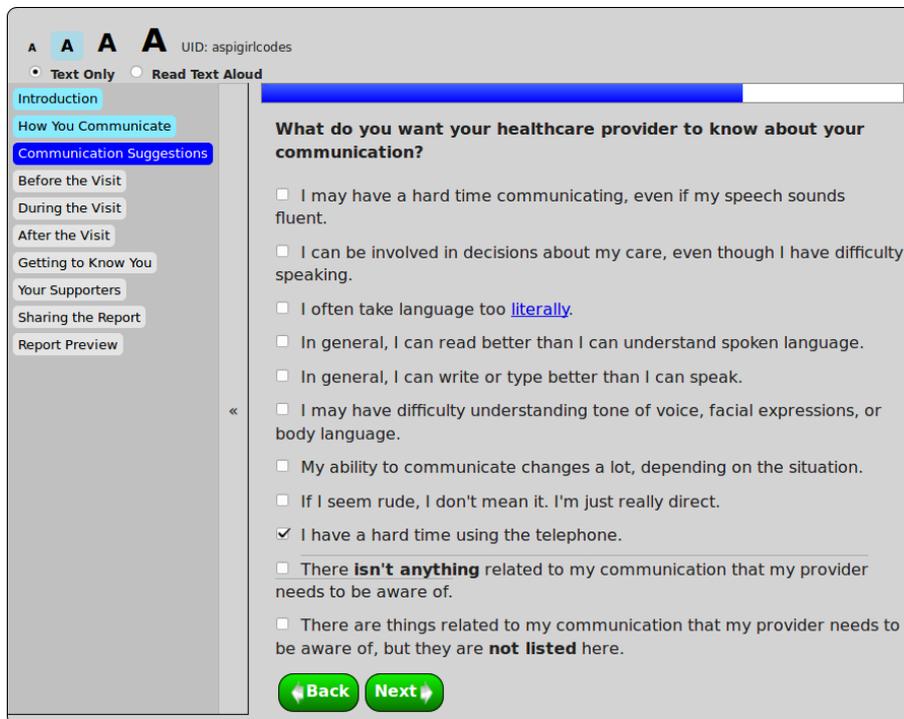


Figure 3.1: Screenshot of the AASPIRE healthcare tool

The resulting tool, of which a screenshot is shown in Figure 3.1, also has a couple of drawbacks. It focuses a lot on detail, resulting in long lists of questions for the user to answer as well as a long

PDF-report for the healthcare provider to read. They do comply to the Web Content Accessibility Guidelines by W3C [64]. But on the other hand, they seem to have not invested much time in classic usability. Although most of the problems covered are not limited to a healthcare setting, the tool lacks flexibility that would enable people to use its results in other settings as well.

### 3.2.2 Module Creation

Based on the research of the AASPIRE Healthcare toolkit and my own (limited) user research, I worked together with Ms. Schwarz from the Psychiatric department of the FMI Hospital to create a list of modules. Six structured modules with a lot of prefilled content were largely based on the healthcare toolkit research. Like in the healthcare toolkit, they would often contain lists of descriptive sentences where the user can choose what applies to their situation. However, we decided to shorten the lists of options in almost all modules. Instead, we would allow the users to add their own options using free text fields. In addition to the structured modules, we added three completely free modules for text, lists and pictures, to give users even more possibilities to provide their own content.

Below you find a list of modules we agreed on, together with their descriptive help texts that will appear on the website.

**General Info Module:** With this module you can introduce yourself. Add a picture if you want, state your pronouns, or disclose your autism. As in all our modules, everything is optional.

**Communication Module:** This module lets you describe your communication preferences in different situations. In addition, you can give others tips on how to communicate effectively with you.

**Do's and Don'ts Module:** With this module, you can provide three quick lists with things people can do, should not do, or should ask you first.

**Medication Module:** Here you can create a table with your medication and when you take how much of it. This can be useful for doctors or caregivers, or just as a reminder for yourself.

**Sensory Module:** This module presents your sensory profile. You can also add other information related to sensory processing.

**Contact Module:** In this module, you can add all kinds of contact data. As always, all fields are optional. You can add contacts with only a phone number or email address, as well as postal addresses and even maps (not available, yet).

**Free Text Module:** Here you can create a custom module containing text and a title. You can use it for any text you want to add that does not have place in our pre-formulated modules.

**Free List Module:** A place to add your own lists. List things you are good at, your hobbies, questions you have prepared for a conversation, or anything else you want to make a list of.

**Free Picture Module:** Upload your own pictures and add a title and a description to them. Sometimes adding an illustration, cartoon or photo helps to bring your message across.



Figure 3.2: Paper prototype of the “General info module”

### Communication

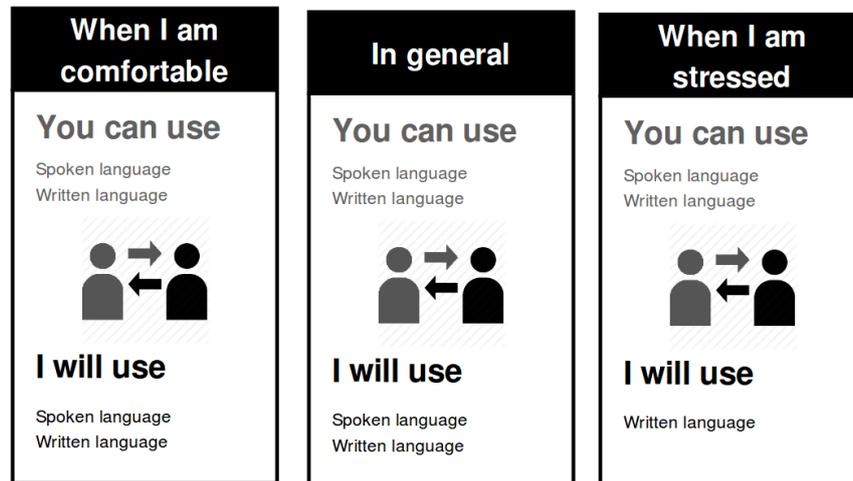


Figure 3.3: Paper prototype of the “Communication module”

### Do's and Don'ts Checklist

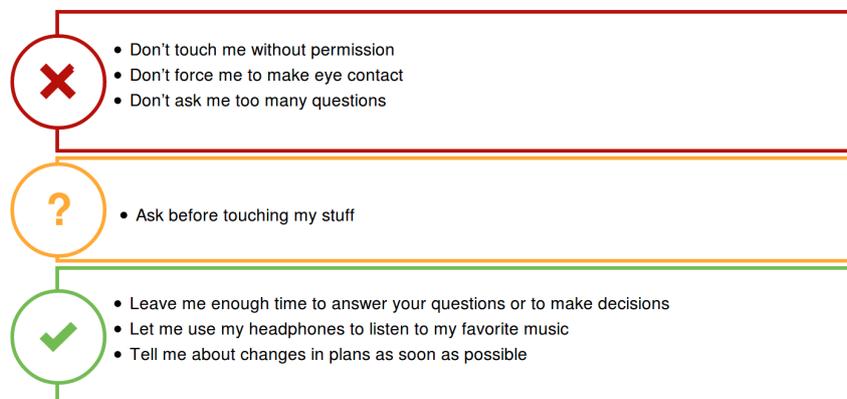


Figure 3.4: Paper prototype of the “Do's and Don'ts module”

**Module: Means of communication**

**Title**

When I am comfortable

|  |   |
|--|---|
| <p><b>Means of communication others may use:</b></p> <p><input type="checkbox"/> Spoken language</p> <p><input type="checkbox"/> Written language</p> <p>Specific text based alternative to speech</p> <p><input checked="" type="checkbox"/> (text-based AAC (Alternative and Augmentative Communication), specific app, ...)</p> <p>Picture-based alternatives to speech</p> <p><input checked="" type="checkbox"/> (picture-based AAC (Alternative and Augmentative Communication), picture boards, drawing)</p> <p><input type="checkbox"/> Official sign language</p> <p><input checked="" type="checkbox"/> Other signs, gestures or behaviors</p> | <p><b>Means of communication you may use:</b></p> <p><input checked="" type="checkbox"/> Spoken language</p> <p><input checked="" type="checkbox"/> Written language</p> <p>Specific text based alternative to speech</p> <p><input checked="" type="checkbox"/> (text-based AAC (Alternative and Augmentative Communication), specific app, ...)</p> <p>Picture-based alternatives to speech</p> <p><input checked="" type="checkbox"/> (picture-based AAC (Alternative and Augmentative Communication), picture boards, drawing)</p> <p><input type="checkbox"/> Official sign language</p> <p><input checked="" type="checkbox"/> Other signs, gestures or behaviors</p> |
|--|---|

Figure 3.5: Paper prototype of the “Communication module” form

### 3.2.3 Paper Prototypes of Modules

After deciding on the modules, I created paper prototypes of most modules and some of the corresponding forms. Examples of those paper prototypes are shown in Figures 3.2, 3.3, 3.4 and 3.5. Those were again discussed with Ms. Schwarz and some improvements were made. As it became clear that forms would become an important part of the project I decided to read “Designing UX: Forms” by Jessica Enders [18] and use it as a reference on form usability throughout the rest of the development.

In her book, Enders advises to follow a four-stage model of how people answer questions that was proposed by cognitive scientist Roger Tourangeau. The stages Tourangeau distinguishes are: comprehension, retrieval, judgment and answering. In each of those stages the user can face challenges, resulting in not being able to properly fill in the form. Enders gives specific advice on wording (use plain language), layout, which field types to use when, how to order questions or options of a question, help texts, error messages and much more. In order to not replicate bad design she advises against looking at other forms online, but rather resort to user research as a resource of information on how to design your form.

## 3.3 Creating Example Infographics

After discussing the paper prototypes of independent modules, I started creating some static HTML pages with example infographics consisting of different modules. I used the CSS framework Bootstrap [12] to be able to quickly design those example pages. The examples created could now easily be shown to possible users to give them an idea of the purpose of the project. Later I could use those static HTML pages as starting point for iteratively developing the interactive parts of the website. The images of those examples can be found in Appendix B.

## Chapter 4

# Technical Implementation and Challenges

### 4.1 Introduction

Discussing every aspect of the technical implementation of the project would be outside the scope of this thesis. Therefore, I chose to give only a short introduction to the main framework used in this project: Django. After that I will discuss two technical challenges in more detail. The representation of the application objects in the database, and the design of the login and registration process. Together they will give the reader a good overview of how the application was implemented.

### 4.2 Django

Django is a Python server side web development framework. Starting 2005, Django is distributed under an open source Berkeley Software Distribution (BSD) license. Its strong points are the out-of-the-box admin interface, its focus on security, and the extensive and well-written documentation [24, 50].

Architecture-wise, Django does not strictly stick to the very common Model-View-Controller (MVC) pattern. Instead it uses a more pipeline-like approach and calls it Model-Template-View Architecture. Requests are guided through several layers of middleware and then dispatched to specific views. Views are request-processors, expected to return a response, that is then guided back through the middleware layers to the server. To do their job as request-processors, views can interface with models and templates. Like in the MVC-pattern, Models are the database interfacing classes. Templates are used for the final representation of data by making use of a templating language [24, 50].

I have worked with Django for web development before and love how it enables me to quickly build an actionable prototype or first version of a website without much hassle. However, using a known framework can only be an advantage if it is also suited for the job. Django is easy to set-up but has a lot of flexibility and can therefore be used for small as well as for larger scale web

applications. As this project uses a lot of forms it can profit from Django's built in features for form validation as well as security. On top of that Django has tools for internationalization and user authentication, both of which will be useful for this project. The Django admin is an easy tool to create, access and edit data both during development and for administrative tasks on the productive system. Based on these features I decided that Django is indeed a suitable framework for this web application.

For this project Django 1.11 LTS, which has extended support until at least 2020 is used with Python 3 and a PostgreSQL database. Django interfaces natively with the most used relational databases such as PostgreSQL, MySQL, and Oracle. I chose to use PostgreSQL in order to be able to use its array datatype to represent "select all that apply" types of data that occur in many modules.

On the front end, the external dependencies used are the CSS framework Bootstrap v4 [12], as well as JQuery in the JavaScript code.

## 4.3 Objects versus Database Tables

### 4.3.1 Problem Description

The piece of logic present in most web frameworks that allows the seamless conversion between database rows and software objects is called Object Relational Mapping (ORM). A good ORM makes smart decisions on when to hit the database and how to optimize queries. Therefore, it allows the programmer to remain mostly in his object oriented programming language world without having to worry too much about the database part. Sometimes, however, fundamental concepts of object oriented programming are not readily transferable to the reality of relational databases. This is the case for all things that have to do with inheritance and polymorphism.

In this project, users will create infographics, which I will also call pages. Each page consists of one or more modules. All modules have some things in common, they have a link to the page they belong to, and a position in this page, for example. But we also defined different types of modules (see Section 3.2.2), and each of those module types will also store different information to the database, have different creation and editing pages, and be rendered differently inside the page.

This modular approach is explicitly desired, as it gives the infographics their needed flexibility. Storing each module separately, instead of only the final resulting page, allows users to edit or exchange single modules from existing infographics, and sort modules to their liking. This way, they can quickly create personalized pages for different situations based on examples that have been created by themselves or others.

To be displayed, each page should have access to all its modules and be able to sort them by position. In object oriented programming we would clearly implement some kind of inheritance for the modules. But how does this translate to relational database tables?

### 4.3.2 Solutions

This is a common problem when combining object oriented programming and relational databases. One solution would be to look into using a non-relational database instead. NoSQL databases are more flexible concerning the data that they store, and therefore can more easily incorporate object oriented concepts.

However, the Django ORM is not constructed for non-relational database interactions. To use Django with a NoSQL database such as MongoDB, one needs to use a third party database back end instead. Because I have never used a non-relational database as a main storage before, I would also need more learning time that I could then not use for other aspects of the project. Therefore, I want to consider other options first.

That this problem can be solved while using relational databases as well, is proven by the fact that the Django ORM offers a couple of solutions, described in the model inheritance section of the documentation [25]. Three styles of inheritance are provided, two of which would be useful in the scenario described above. The third one, the use of so called proxy models, enables only different Python level behavior on models with the same set of fields. In this project, each module will have a clearly distinct set of fields, with only some fields in common, so this type of inheritance is not applicable.

That leaves two possibilities to look into in more detail: Abstract base classes and multi-table inheritance.

#### Abstract Base Class

In the case of using an abstract base class, this class will exist solely in the code. You will never be able to instantiate base class objects, and the base class will not have its own table in the database. That is why it is called abstract. It exists as a means to collect those parts (both data/fields, and logic) that different models have in common to keep your code more DRY (“Don’t repeat yourself”).

In this project, this would mean that we have a base module containing the foreign key field to page as well as the position field and possibly other fields all modules might have in common. If one would define methods that are the same for all modules, for example a method to increase or decrease the position of a module, those would be part of the base module class. Specific modules would inherit from the base module class, and define their own additional fields and methods. If necessary, some modules could overwrite methods of the base module class.

Each module would also be represented by one table in the database, with columns for both the fields defined in the abstract base module and in the specific module. So all modules would have a position column as well as a page id column representing the foreign key relationship of the module to page, as well as all the columns for the fields defined in that specific module.

This also means that if the page wants to get all its modules, sorted by position, it has to query each module table and then sort the entries in memory. This is a drawback of the approach, and could get problematic if there would be really large amounts of all kinds of modules in one page.

The advantage however is that all the information on one module is in a single table. This will not be the case for multi-table inheritance, as the name already suggests and as I will discuss next.

### Multi-Table Inheritance

In multi-table inheritance, each model class in Python corresponds to a table in the database. Relationships between parents and children in the inheritance hierarchy are represented on the database level by links introduced as hidden one-to-one fields by the ORM. This makes it easy to navigate between the objects in a way similar to explicit relationships. However, behind the scenes, new queries are made and tables are joined.

For this project, using multi-table inheritance would mean that the basemodule would have its own table with the foreign key field to page and the module position. Therefore, the page could easily request all its modules sorted by position. However, each time one would want to know something about the actual content of the module, a new query would be necessary.

### 4.3.3 Proposed Model Structure

I chose to use the abstract base class solution for this project. Using multi-table inheritance is discouraged by most experienced Django developers unless you have a good reason to do so [30]. I had no such reason. Moreover, I expect pages to have a limited number of modules, so using the abstract base class approach would be no problem even though I have to sort the modules in memory.

This resulted in the model structure depicted in Figure 4.1. Each module inherits from the abstract “Module” class, which defines the common fields for the position of the module and the foreign key to the page.

The “Page” class is connected by a foreign key to the user who created the page. It has a title, and knows how many modules it contains. Further fields indicate whether this is an example page and whether it is publicly visible. The fields `token`, `token count` and `token timestamp` enable the creation of a secret link for sharing the page. The exact mechanism of how this is done is explained later in this chapter in Section 4.4.4. The “Page” class also contains methods to retrieve all related modules and sort them by their position.

The aim of each of the nine modules is described in Section 3.2.2. In many of the modules there are fields of the form `<fieldname>_choices` and `<fieldname>_free`. The `_choices` field represents a list of statements where the user can select all statements that apply to them through ticking checkboxes. The `_free` field enables the user to add statements of their own. I often use this mechanism to provide both enough guidance and freedom for the entries.

Both fields rely on the possibility of PostgreSQL to define columns of a table as variable-length arrays, and the availability of matching fieldtypes in the Django ORM. They allow me to easily store the multiple statements the user has chosen or entered in one table column as a one dimensional array of varying character entries. The `_choices` fields on top of that make use of the ability of Django to relate a set of predefined choices to a character field (as well as to other field types). Entries of `_choices` and `_free` fields are kept separate for easier representation in forms.

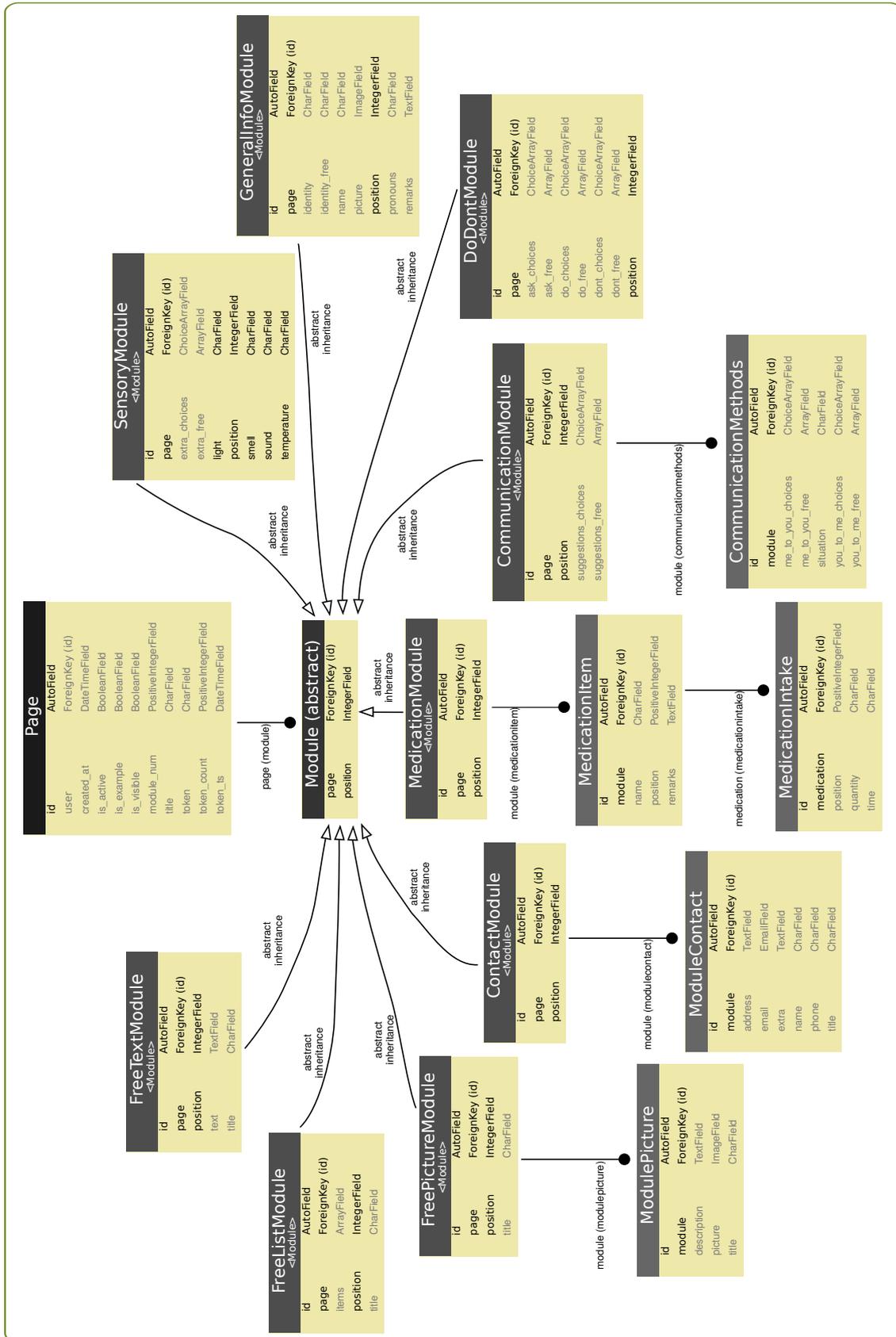


Figure 4.1: Graphical representation of models

Some modules can contain multiple items of a related entity. The contact module, for example, can contain contact data of multiple contacts. The free picture module can contain multiple pictures, each with their title and description. The communication module can contain preferred communication methods for different situations. And the medication module can contain several medication items, which in turn can have information for multiple medication intakes each. Those entities are stored in separate tables and relate to the original module through foreign key (many-to-one) relationships.

## 4.4 User Authentication

### 4.4.1 User Authentication in Django

Django comes with a user authentication system, that is both extensive and flexible. It can handle users, groups and permissions, contains a password hashing system, and different helper tools for registering and logging in users, as well as restricting content access.

The default Django user object has an obligatory and unique username, a password, email, `first_name` and `last_name`. Like the username, the password is obligatory, but you can set a so called unusable password (which will never validate) as well. It is always stored in its hashed form. All other fields are optional.

Django provides a set of views and forms for logging in and out as well as for password management. Keeping the logic separate between views and forms allows for easier customization. There are views for login, logout, password change (when knowing the old password) and password reset (with a password reset link).

On top of that, there are wrappers and mixins to add to your existing views to make sure they are only accessible by logged in users, or by users that are logged in and fulfill additional requirements. The latter makes it easy to give users only access to objects created by themselves, or can be used to check extra permissions the user requires [25].

### 4.4.2 Registration and Login Design for this Project

#### Design Considerations

For this project, I want to use Django's authentication system wherever possible, but I will also need to make adaptations to it. One thing I will certainly have to implement myself is the registration part, as it is not included in Django.

A first question to answer is: What do I need to know about the user to create their account? In this case, user accounts are purely for the users themselves to manage their created pages. I will never display their username publicly. Therefore, it makes sense to have no separate usernames, but just use the users' email address. Knowing the user's email address will be useful as it allows me to send them a password reset link when they forget their password. Apart from the email address and password, there is really nothing else I need to know.

The next thing I need to think about is how to design the registration. In theory I could let the user play around and create their first page with some modules already, and only in the end ask

them if they want to create an account to save the result. This technique is called delayed or gradual engagement, and it is used to lower the barrier for using a product.

However, it also has some disadvantages. The user might be aware from the start that at some point they will need an account, and worry about where and when they will get to create it. Did they miss the button? Will their work be lost? What information will they need for the account? Will they suddenly hit a pay-wall or be required to provide information they do not want to provide in order to open an account? What if they get overwhelmed or need to do something else in the middle of the creation process? I do not want to burden the user with so many uncertainties. On the contrary, I want to communicate clearly and up front.

Therefore, I will let the user create their user account at the beginning of the page creation process. The next dilemma pops up when trying to make this idea more concrete. On the one hand, I want to verify the user's email address, on the other hand, I do not want them to have to switch contexts again. They just let me know that they want to create something on my page, so I do not want to send them around to their email program before they can do that. I do not even want them to have to think about a password yet. If I want them to get started as quickly as possible, I can just ask them for their email address and let them move on, sending the verification email in the background and letting them set their password at a later time.

But also this setup can be confusing for the user. I will have to clearly communicate that they have created an account already, although they have only just given me their email address, and that they can move on now without being scared to lose what they will be creating, even if they enter the password at a later point in time.

In a bigger project these are different scenarios that should be tested out with a group of target users. However, this would be beyond the scope of a bachelors project. Instead I have to make an educated guess, and carefully look at how the users work with it during the general user testing.

### **Proposed Design**

The default Django user model uses two separate fields for username and email address. For this project, however, I want users to use their email address as a username. Instead of creating a whole new User model, duplicating much of the functionality Django already has, I can still use Django's user model with a few alterations. I add email validation to the username field and change the field's label in all login related forms. I also copy the username, which is now a validated email address, to the email field of the user during the registration process. This way, views and forms sending emails can rely on the email address being in the email field, and I do not have to change all of them to look at the username field instead.

Three types of users might be landing on the different login and registration pages: new users, users with an unverified account and no password yet, and users that have a password and verified email address. I need to make sure that each of them has the possibility to authenticate and gets the right email messages in their inbox.

A new user can start creating a page right away after registration (entering their email address). Although the second time they come back (as a user with an unverified account), I want them to set their password first, before they can continue.

Also, users with an unverified account should be able to use the forgotten password process as well as the registration process to get the verification link sent to their inbox again. After all they might not remember whether or not they already set a password.

After logging in or registering, users that have not created any pages yet should be redirected to the page creation form, whereas those who already have one or more pages, should be redirected to a list of their pages, from where they can edit and manage existing pages, or decide to create a new one.

The resulting registration and login process is illustrated in Figure 4.2. Each column represents a page of the web app. The numbers 1,2 and 3 represent the three types of users. The diagram depicts what happens when they access the pages through get and post. Horizontal arrows represent navigation or redirection between pages. Vertical arrows between get and post represent the action of submitting a form on a page.

### 4.4.3 Password Reset Links in Django

As mentioned in Section 4.4.1, Django's user authentication system comes with a set of views that allow the user to reset their password. We might be able to use this mechanism not only for password reset, but also for email verification, or even for creating secret links through which users can share their pages. In order to understand how this can be done, I want to first have a look at how exactly the mechanism works.

#### The Password Reset Mechanism in General

Almost all websites that use a password to authenticate users also offer a way of recovering from the situation where the user does not remember their password. Most do so by sending a one time link to the users email address. The user can then authenticate themselves by clicking the link and subsequently setting a new password. To make this possible the link contains two key elements. A user-id, which is a unique identifier for this user, and an authentication token. The authentication token is seemingly random and is often chosen to be relatively long to improve security. Ideally, this token should not be valid indefinitely, but should expire after an appropriate amount of time depending on security and usability requirements.

Based on the above description, a straightforward implementation of this feature is:

1. When a user has forgotten their password, they have to enter their email address to start the recovery process.
2. Based on their email address, the system can find the user in the database. It can then store the current timestamp and a randomly generated authentication token in the respective columns of the user table. A permanent unique identifier of this user, together with the authentication token, is used to generate the reset link that is sent to the users email address.
3. When the user clicks the link, the app can retrieve both the user id and the authentication token from it, and check if it matches the one in the database. Also, the stored timestamp has to be compared with the current date and time to verify the token hasnt expired yet. If all seems right the user is allowed to set a new password. The value of the users authentication token in the database is set to Null, so that the link becomes invalid.

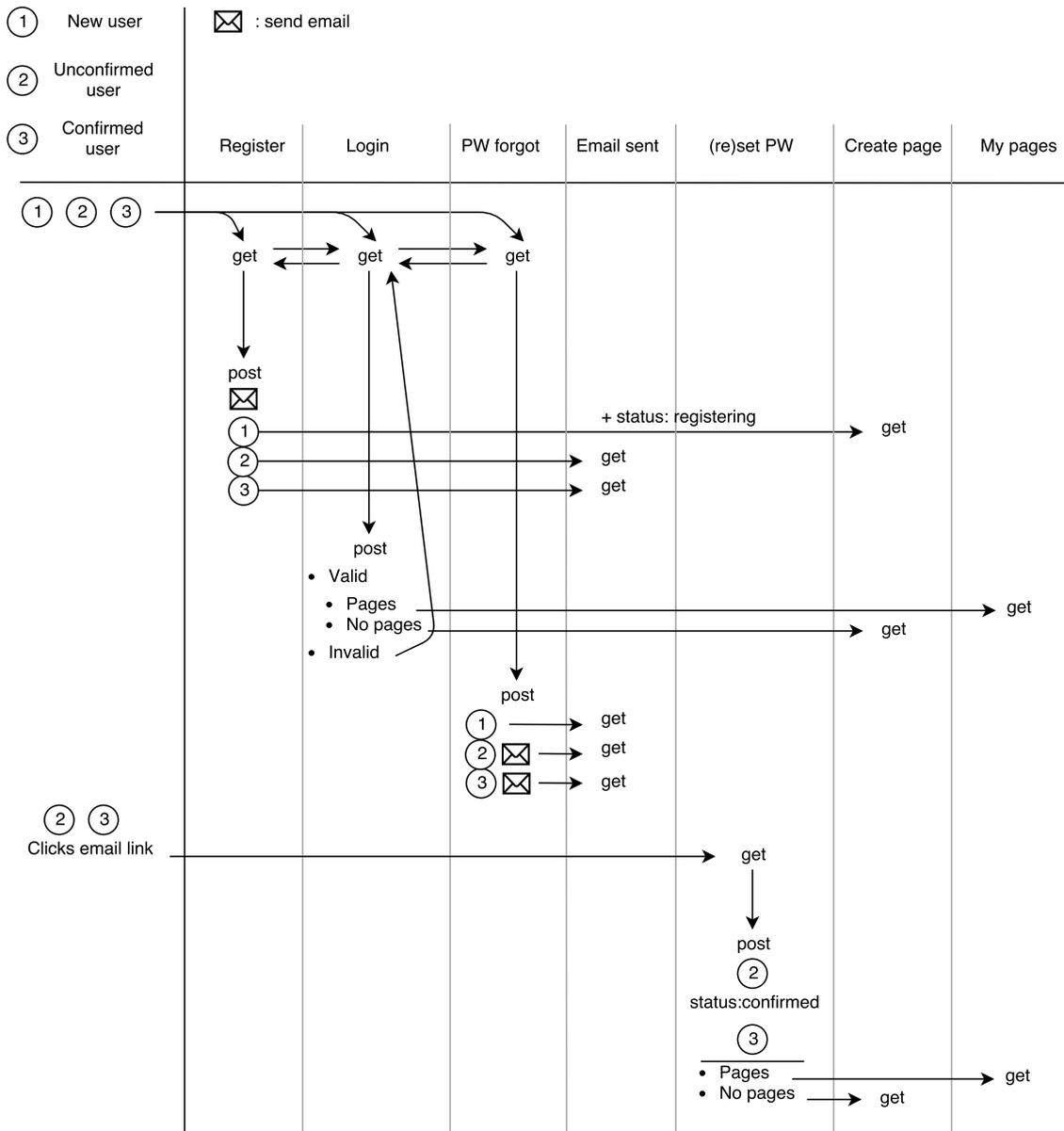


Figure 4.2: Overview of the registration and login process. Each column represents a page of the web app. The numbers 1,2 and 3 represent the three types of users. The diagram depicts what happens when they access the pages through get and post. Horizontal arrows represent navigation or redirection between pages. Vertical arrows between get and post represent the action of submitting a form on a page.

### Django's Implementation of Password Reset

Django however uses a different approach that does not require storing an authentication token or timestamp to the database. It goes as follows:

Like in the previous approach, the password reset link contains both a unique user identifier and a token. But this time, the authentication token is different. It consists of two parts. The first part is just a Base-36 encoded timestamp. The second part is a hash calculated over the following values: the user's primary key (the unique identifier used), their (hashed) password, their last login timestamp, and the current timestamp. A secret key is used to calculate the hash value, and it should not be reproducible as long as this key isn't compromised.

To authenticate the user based on the password reset link, Django first splits the token into its two parts and decodes the timestamp. At this point, Django has access to all the information that was used to create the hashed part of the token, being the user's id (which is part of the link), their (hashed) password and last login timestamp (retrieved from the database), as well as the timestamp just retrieved from the first part of the token. It can then use its secret key to recreate the hashed part of the token and check whether it is equal to the one from the link. The same timestamp can also be used to verify that the token has not expired yet. When all is fine, the user can reset their password. Due to the password reset, both the login timestamp and the password will change, automatically invalidating any previously generated links [25, 26].

#### 4.4.4 Using Django's Password Reset Mechanism for Different Purposes

##### Registration and Email Verification Link

As described in Section 4.4.2, in this project, the user's email address will be used as a way of identification (instead of a username), as well as for password recovery. It is, therefore, sensible to check whether the user does have access to the provided email address as part of the registration process. This can easily be done using Django's password reset mechanism with just a few adaptations.

The registration workflow initially only requires an email address from the user, and creates an account with an unusable password and an `email_confirmed` boolean set to false. In a next step, after clicking the registration link in the email, the user can set their password, and their status will change to `email_confirmed = True`. Just like the user's password and last login timestamp, the `email_confirmed` boolean is a piece of information about the user that changes after the link has been used. Therefore, a token generator for email verification can be created by simply adding this boolean to the hashed information of the default token generator described in Section 4.4.3 [26, 21].

##### Sharing Pages Using a Link

An important feature of the webapp is the possibility for the user to generate a unique link that they can share, and which gives others read-only access to a specific page they created. At any time the user can choose to generate a new link, invalidating old links in the process.

Instead of generating a unique random token (and having to check uniqueness of this token to all

---

previously generated tokens) and storing that in the page's table in the database, I can make use of Django's password reset mechanism. This time, the link will contain the page's unique identifier instead of that of the user. Besides the page's primary key, the token generated for this purpose will contain the link creation timestamp, and a token count. When the user generates a new link, the link creation timestamp, as well as the number of tokens that were generated for this page (the token count) will change, invalidating old links.

However, as I do not want to regenerate the token each time it is shown, this time it is probably useful to store it in the page's table. Therefore, a part of the appeal of the original mechanism is lost. The remaining positives are the fact that I can use a mechanism that I am already familiar with, and that it is easier to make a unique hashed value containing a timestamp than creating a unique random value.

# Chapter 5

## Validation

The web application was iteratively developed and tested by various stakeholders. An overview of the different versions and test persons can be found in Table 5.1. A more detailed description of the most important findings is given below.

Table 5.1: Iterative development and testing, an overview of versions and test persons

| Version | Content   | Testing   |
|---------|---|---|
| 0.1     | Statical homepage and examples  | Ms. Schwarz   |
| 0.2     | First interactive version,<br>* user can create modules   | Ms. Schwarz,<br>M. (Technical user)                           |
| 0.3     | Editing deleting and sorting modules  | M. (Technical user)   |
| 0.4     | User accounts and page management<br>* user can register using their email address<br>* user can login, logout, reset and change their password<br>* user can manage pages:<br>changing visibility and generating links for sharing | G. (Normal user),<br>C. (Normal user),<br>N. (Technical user) |
| 0.4.1   | Version 4 with improved usability   | T. (Normal user),<br>S. (Normal user)                         |
| 0.5     | Final version<br>* user can delete a page<br>* user can clone examples as starting point<br>* improved general info module<br>* improved registration   | Ms. Schwarz,<br>J. (Normal user)                              |

### 5.1 Version 0.1

Version 0.1 was a purely static website. Besides a homepage, it contained three examples of infographics, as described in section 3.3 and depicted in Appendix B. It was reviewed by Ms. Schwarz. She noticed that dynamic navigation bar highlighting was still missing. The infographic examples matched her expectations. She also mentioned an additional possible use case that would be interesting for her as a psychologist. She would like to integrate the infographics in her

therapeutic practice for autistic people after diagnosis, in order to help them discuss and improve their self-evaluation.

## 5.2 Version 0.2 and 0.3

In the next two versions users could start creating their own infographics. However, the notion of a user account itself was still not implemented. Due to this, the process of interaction with the web app was still fractional and a bit awkward. Therefore, I decided to test those versions with someone who is acquainted with software development. I call this type of user a technical user. This testing was very useful to identify some technical bugs.

## 5.3 Version 0.4

Version 0.4 was the first interactive one to be tested by non-technical users. The focus of testing therefore also shifted from finding technical bugs to usability. Test users were both autistic people and caretakers of autistic children. They were either observed while interacting with the website as in classical user testing, or performed the task of creating an infographic for a specific occasion on their own and reported on their experience in writing.

Several usability issues were discovered this way. Very often the same issue was mentioned by multiple test persons. Their feedback led to a number of usability improvements implemented in Version 0.4.1. Where this was the case both the issue and the improvement are described in this section. Other issues were delayed to be fixed in Version 0.5. For those, only the issue is described in this section, whereas implementation details of the improvement are described in Section 5.5

### 5.3.1 Call to Action Button in the Navigation Bar

Originally the link to start building an infographic was highlighted in the navigation bar, as it was the desired place to direct the user to (see Figure 5.1). The highlighting as a green outlined call to action button was similar to call to action buttons below the navigation bar on the homepage and different from the highlighting of the page the user currently is at (indicated by white instead of gray text in the navigation bar). Despite this, users felt like this green outline indicated the place where they currently were inside the web application. Instead of leading users towards the page where they could start doing something, the highlighting was just confusing them. Therefore, it was removed.



Figure 5.1: The navbar in Version 0.4 with the call to action button and items that are dead links, and the improved one in Version 0.4.1

### 5.3.2 The purpose of the Web Application

Users found it hard to tell what the aim of the web app was by looking at the homepage. The homepage indicated which steps were needed to create an infographic, but not what the idea behind it was. To find out more, users tried to click links in the navigation bar, such as “How it works”, and “FAQ”, as well as “Examples”. However, the first two of those navigation bar items had no information behind them (yet).

#### The Idea



Figure 5.2: Additional information on the homepage about the purpose of the webapp.

To improve this usability issue, I changed three things: I removed dead links from the navigation bar (see Figure 5.1) and I added more information about the purpose of the web app both on the homepage itself (see Figure 5.2), and in the FAQ section. The latter now has the answer to three questions: “Who can make an autism infographic?”, “What situations can autism infographics be used for?” and “Can I make more than one?”.

### 5.3.3 Previewing the Work During Creation

Often while you are working on something, you want to see what the result will look like, especially if the tool you are using is new to you. Originally, it was not possible to preview the end result of the infographic during the creation process. This bothered users. The method of clicking the “I am finished” button, just to see the preview, and then return to the editing page is not at all user friendly. Therefore, a preview button was added that could be easily used during the creation process. While creating the infographic users see a list of modules they have already created. Each item is originally collapsed but can be clicked to see a preview of each module separately. In Version 0.4 it was however not possible to see a preview of the infographic as a whole from this position. The preview button on the top right was added in Version 0.4.1

Figure 5.3 shows the context and the improvement.

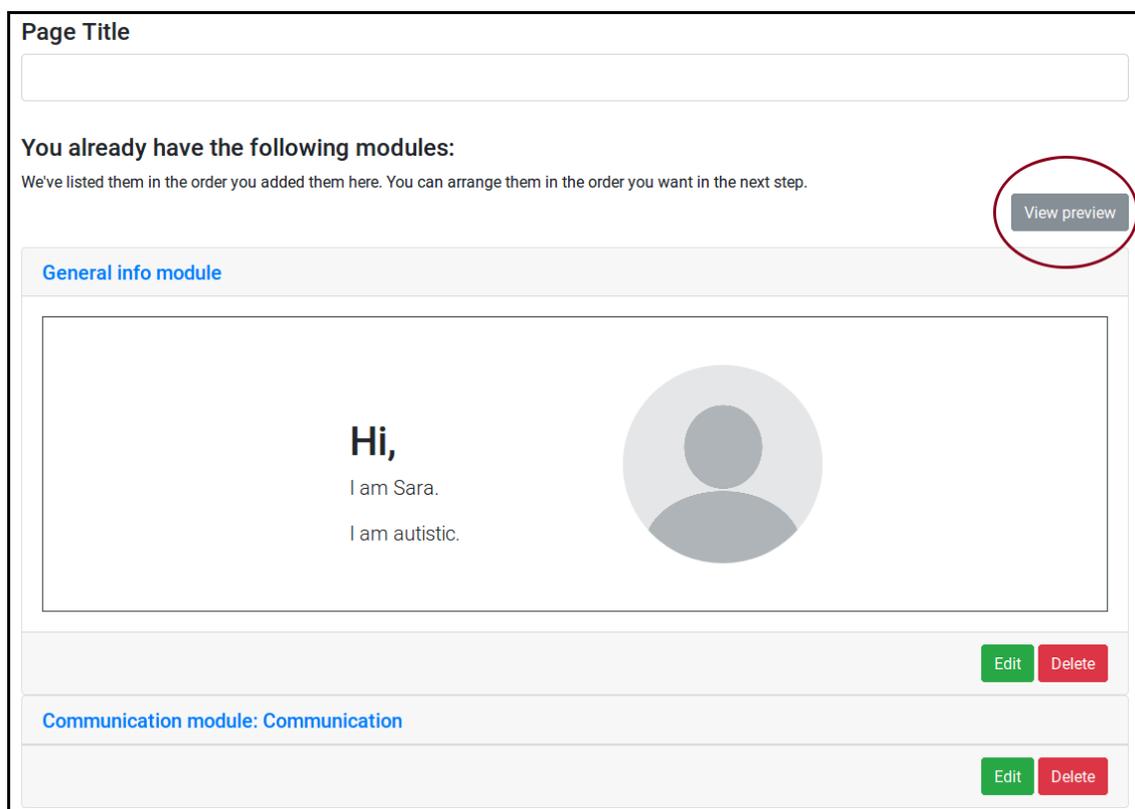


Figure 5.3: While creating the infographic users see a list of modules they have already created. The Preview button on the top right (circled in red) was added in Version 0.4.1

**General info module**  
With this module you can introduce yourself. Add a picture if you want to, state your pronouns or disclose your autism. As in all our modules, all of this is optional.

**Communication module**  
This module lets you describe your communication preferences in different situations. In addition you can give others tips on how to communicate effectively with you.

**Do's and Don'ts module**  
With this module, you can provide 3 quick lists with things people can do, shouldn't do, or should ask you first.

**Medication module**  
Here you can create a table with your medication and when you take how much of it. This can be useful for doctors or caregivers, or just as a reminder for yourself.

**Sensory module**  
This module presents your sensory profile. You can also add other information related to sensory processing.

**Contact module**  
In this module you can add all kinds of contact data. As always, all fields are optional, so you can add contacts with only a phone number or email address, as well as postal addresses and even maps(not available yet).

**Free text module**  
Here you can create a custom module containing text and a title. You can use it for any text you want to add that doesn't have place in our pre-formulated modules.

**Free list module**  
A place to add your own lists. List things you are good at, your hobbies, questions you have prepared for a conversation, or anything else you want to make a list of.

**Free picture module**  
Upload your own pictures and add a title and a description to them. Sometimes adding an illustration, cartoon or photo helps to bring your message across.

## V 0.4

**General info module**  
With this module you can introduce yourself. Add a picture if you want to, state your pronouns or disclose your autism. As in all our modules, all of this is optional.

**Communication module**  
This module lets you describe your communication preferences in different situations. In addition you can give others tips on how to communicate effectively with you.

**Do's and Don'ts module**  
With this module, you can provide 3 quick lists with things people can do, shouldn't do, or should ask you first.

## V 0.4.1

[Create module](#)

[Create module](#)

Figure 5.4: List of modules and submit button: Version 0.4 is shown on the left side. The frame on the right shows the improvement made in Version 0.4.1.

### 5.3.4 Scrolling Needed to Find a button

Also during the infographic creation process users select a module they want to work on from a list of modules represented together with a small help text through radio buttons. They then confirm that they want to create this module by clicking the “create module” button, which is below the list. As the list is quite long, users often have to scroll down to get to this confirmation button, and this can be confusing. To improve usability, I therefore added some JavaScript to make a submit button appear right next to the item the user just selected.

Figure 5.4 shows the long list of options with the submit button only at the bottom as it was in Version 0.4 on the left, as well as the improvement of Version 0.4.1, where the button is displayed right next to the selected module, to prevent that the user has to scroll down to find it.

### 5.3.5 Registration Process

The registration process implemented employs some kind of gradual engagement. Users can start creating their first infographic right after they enter their email address. While they are doing so, an email is sent to them to confirm the email address and set a password. However those things do not need to be done immediately. A message at the top of the infographic creation page that comes up after entering the email address explains this to the user. However, this message was often missed by the test users, or it was confusing to them, whether they had to act upon it or not.

Making the message appear before the page with the form is displayed and tweaking the message content could easily improve this process. However, at this point, it was not yet clear to me whether it was just the message that was confusing, or the concept of gradual engagement. I decided to not yet change anything to the registration process, and have a closer look with the next round of test users.

### 5.3.6 Cloning Examples as a Starting Point

After looking at the examples, starting to create their own infographic from scratch seemed a bit daunting to some test users. They would have liked to be able to clone one of the example infographics and then adapt it to their own needs. Although this wish is understandable, it is rather a new feature than simple usability improvement. I therefore decided not to implement this for Version 0.4.1 yet, but to keep it in mind for the next version.

## 5.4 Version 0.4.1

The improvements discussed above were deployed as Version 0.4.1 and then again tested by different users using the same procedure as in the previous version. Confusion during the registration process was still an issue for some testers, and also the feature request for cloning examples was repeated. These two improvements would, therefore, be the main focus for the final Version 0.5.

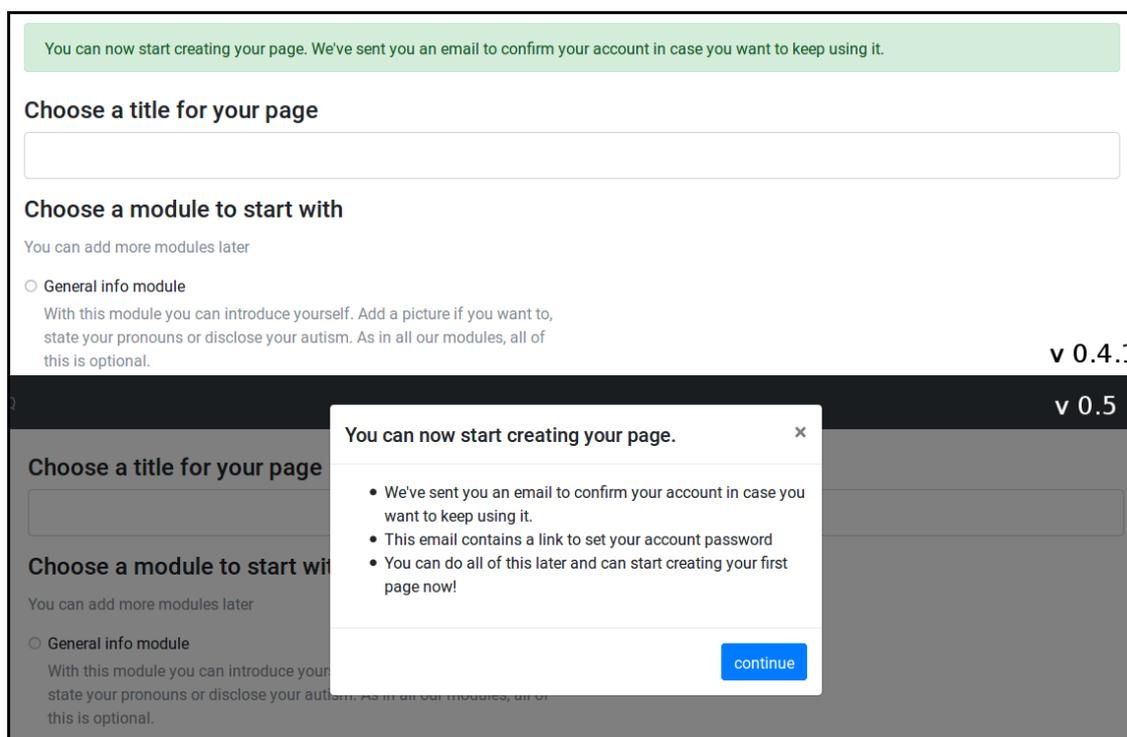


Figure 5.5: Top: The message at the top of the form in Version 0.4.1 was easily overlooked by users. Bottom: Improved message in Version 0.5.

## 5.5 Version 0.5

Version 0.5 had a few new features, the most important of which is the ability to clone an example as a starting point for creating a page. Also, the information about the registration confirmation email is made more prominent in this version. In previous versions users tended to overlook the message and were confused because of this. When shown the message, users found the registration process much more clear. From this, I concluded that the message should be displayed in a way users cannot overlook it. This improvement is shown in Figure 5.5.

All changes for this version were highly appreciated by the test users and the end result was approved by Ms. Schwarz.

# Chapter 6

## Conclusion

### 6.1 Discussion

The aim of this project was to lower the barriers of communication about autism in different situations. Ultimately, I hope that through more and better communication, more understanding and acceptance can be reached, without putting autistic people in unnecessary stressful situations.

The practical implementation is based on two existing tools. An analog one, the generic autism id card, and a digital one, the AASPIRE autism and health tool. By combining the ideas of both tools, I was able to create a new tool that is more versatile and flexible. It can be used in a wider range of situations than the autism and health tool, and it can be more easily adapted to the individuality of each autistic person than the autism id card.

Through the modularity of the infographics, and the free text options in each module, the possibilities for personalization are endless. At the same time, the preformulated statements, and the example infographics that can be used as a starting point for customization, provide enough guidance for users not to get lost.

I chose to create the tool in form of a web-application, making it platform independent. The possibility to progressively add offline capabilities was planned for, but I was not yet able to implement anything in that direction. Using Django as a back end framework enabled me to quickly create a robust, production ready proof of concept with a lot of functionality. Whereas the iterative approach with user testing throughout enabled me to considerably improve the usability of the application.

On a personal level, being an autistic developer in a one person team on this project was not always easy. Although I love making software, it is undeniable that some parts of the software development process are incredibly hard for me. Luckily, I could get help when needed. Looking back on this experience, I am convinced, that if we can empower autistic and neurodivergent software developers in a non-patronizing way, great projects can arise, improving the life quality of many people.

## 6.2 Future work

Although the end result of this project is a viable product with extensive functionality, there are also some things left in the backlog, waiting to be implemented.

As already mentioned in the previous section, offline functionality is one of those things. I got acquainted with service workers through Tal Aters book on building progressive web apps [2], and wrote a tutorial on how to add them to a Django project (see Appendix A), but I did not yet add any offline functionality to the actual project web application. Doing this will allow users to save their infographics to their phone and showing them to others when needed, even without an internet connection. This would be a second important way of how the infographics can be shared, next to the already existing secret link.

Full multilingual support is another point on the bucket list. For now, all preformulated text can be translated, and is available in both English and German at the moment. The text entered by the user however, is not translatable. I would like to add a possibility for the user to enter those personal texts in multiple languages. This way, they can create infographics that are fully multilingual. There are several Django packages available that serve exactly this purpose. An overview and comparison of the possibilities was given by Raphael Michel in his recent talk on “Data internationalization in Django” at Djangocon Europe 2017 [36].

A further important part of the web application that has not received enough attention yet is accessibility. Although Bootstrap can be used to build accessible websites, it is not accessible by default. Bootstrap’s default color palette for example, has insufficient color contrast to comply with the WCAG accessibility guidelines[64, 12]. Regardless of whether or not one uses Bootstrap or any other CSS framework, the accessibility of the resulting website will be highly dependent on how much attention it has gotten during the implementation. I have not been able to focus on this topic well enough throughout this project. The main reason for that is that I simply know too little about it. Working on this will therefore not only improve this web application, but also my abilities as a web developer.

All three of the above topics have been on my list of desirable features from the beginning of the project. The importance of the example infographics on the other hand is something I did not have on my radar. Only through user testing did I find out that these actually are a key feature to make the application user friendly. Therefore, I would like to improve the example page to be able to accommodate more examples, in different categories, with a short explanation, and searchable by keywords.

Last but not least there are a few things remaining to make the application ready for a possible public go-live. It still needs more careful handling of uploaded images for example. Both from a security and from a storage capacity point of view. Although some logging is already done, this would need to be improved, and monitoring of critical warnings should be added. On the more mundane side of things I would need to rethink the application’s name, find a domain and change to a proper email service provider.

All in all, both me and the people of the psychiatric department of the FMI are happy with the results achieved during this project. I hope that some of the above mentioned improvements can

be implemented, and that the web application will be made publicly available, so that autistic people, parents, friends and therapists can make use of it to improve the general understanding of autism and the life quality of autistic people.

# Appendix A

## Tutorial: Adding Service Workers to your Django App

### A.1 Introduction

In this tutorial you will learn how to add offline functionality to a Django web-app using a service worker. A service worker is a script written in JavaScript that is run by the browser. However, instead of running inside your page like your normal client-side code, it runs in the background. From this position, the service worker can interact both with the pages it controls, and the server (although it does not have direct access to any single pages DOM). In particular, a service worker can intercept requests from a page to the server and do something with them. It could cache certain responses and return them from the cache when another request matching those responses comes in, but it can also send different requests, or read the responses before returning them back to the page.

The unique possibilities of service workers make them very powerful. To fully leverage this power, a whole ecosystem of APIs is being build around service workers, including data storage, background sync, and push notifications. Those allow websites to evolve from a simple web page to a full fledged progressive web app that can do everything a native app can.

Despite the possibilities, the aim of this tutorial is not to build a progressive web app with lots of bells and whistles. That would definitely be too much to cover in one go. Things like background sync, push messages and IndexedDB (a client-side data storage) will not be covered here. Instead the tutorial will focus on the basics of adding a simple service worker to an existing Django project and discuss things like caching strategies and service worker life cycle. If you want to go further after that, you will find useful resources in the what's next section at the end of this tutorial.

**Note:** Most of this tutorial is inspired by Raphael Merx's talk on offline Django with Service workers at Pycon Australia 2017 [35] and Tal Ater's book Building Progressive Web Apps [2].

## A.2 Prerequisites

The starting point of this tutorial is the end of the official Django tutorial (<https://docs.djangoproject.com/en/1.11/intro/tutorial01/>). A repository with the codebase to start from is provided. Download instructions are further down in this section. The Django tutorial is written for Django 1.11 and Python 3. It consists of 7 parts in which you learn how to create a simple poll app.

The app is based on two data models. The question model has a publication date and a question text. The choice model on the other hand has a foreign key relationship with question. This means that one question can relate to multiple choices, but each choice can only belong to one question. Choices have a choice text and keep track of the number of times the choice has been chosen.

There are also three main views: one displays a list of questions, the next displays one question with its choices in a form for voting, and the last one shows the result of the votes for one question. There are no custom views for creating and publishing questions. Instead, the Django admin is used for this.

This tutorial, however, will only add offline functionality to the custom views, and leave the Django admin untouched. If you don't want to do the Django tutorial (again), the final state of the Django tutorial, and thus the beginning state of this tutorial is tagged 'start' in the following GitHub repository: <https://github.com/aspigirlcodes/django-serviceworker-tutorial.git>. This means you can access it as follows:

---

shell

```
$ git clone https://github.com/aspigirlcodes/django-serviceworker-tutorial.git
$ cd django-serviceworker-tutorial/
$ git checkout tags/start -b mytutorial
```

---

**Note:** You still have to create a virtual environment and install Django 1.11 after cloning the repository.

---

```
$ python3 -m venv service_worker
$ source service_worker/bin/activate
$ pip install Django==1.11
```

---

Then set up your project by doing the initial migrations and creating a superuser

---

```
$ python manage.py migrate
$ django-admin createsuperuser
```

---

If you have finished the Django tutorial, you will have enough knowledge of Python and Django to follow along with this tutorial. As service workers are implemented in JavaScript, some experience with JavaScript is useful as well. However, you do not need to be a JavaScript expert. If you have followed a beginner's tutorial about JavaScript or added some JavaScript to a web page by Google, trial and error, you should be good to go. One thing you might want to have a look at if you are not familiar with them already are JavaScript promises, as they are used a lot in service workers.

You will also need a browser with developer tools for service workers. The browsers Firefox and Chrome both have this.

## A.3 Preparation: Improve Django Templates to Output a Propper HTML Page

The Django tutorial sticks to the bare minimum when it comes to the templates HTML content. Html, head and body tags are not included. This is perfectly acceptable for a back end framework introduction tutorial, but now that you will start working on the front end more, it is a good time to fix this first.

First, create a new file named `base.html` in `polls/templates/polls`, and put the following content in it.

---

polls/templates/polls/base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
      shrink-to-fit=no">
    <title>{% block title %}{% endblock title %}</title>

    <link rel="stylesheet" type="text/css"
      href="{% static "polls/style.css" %}" />
  </head>
  <body>
    {% block content %}
    {% endblock content %}
    {% block scripts %}
    {% endblock scripts %}
  </body>
</html>
```

---

Then, change the three existing templates. Note that you can delete the stylesheet link in `index.html`, as it is included in the above `base.html` template.

---

polls/templates/polls/index.html

```
{% extends "polls/base.html" %}

{% block title %} Polls List {% endblock title %}
{% block content %}
  {% if latest_question_list %}
    ...
  {% endif %}
{% endblock content %}
```

---

---

polls/templates/polls/detail.html

```
{% extends "polls/base.html" %}

{% block title %} Answer poll {% endblock title %}

{% block content %}
    <h1>...
    ...
    </form>
{% endblock content %}
```

---

polls/templates/polls/results.html

```
{% extends "polls/base.html" %}

{% block title %} Poll results {% endblock title %}

{% block content %}
    <h1>...
    ...
    ....</a>
{% endblock content %}
```

---

Although this is not absolutely necessary, you can also add the following extra rule to `style.css` so that the html and body elements fill the whole page. By this, the image, set to be in the lower right corner of the body element, will be at the bottom of the page, no matter how much content the page has.

---

polls/static/polls/style.css

```
html, body{
    height:100% !important;
}
...
```

---

That was it for the first step! You are now ready to begin working on our first service worker script.

## A.4 Adding a Service Worker to Return a Simple Offline Message.

### A.4.1 Registering Your First Service Worker

The service worker logic will live in a separate JavaScript file called `serviceworker.js`. The location of this file as seen by the browser is important for the scope the service worker will have. The maximum scope of the service worker is limited by the fact that it can only control files that are below the `serviceworker.js` file in the URL path hierarchy. JavaScript files are static files and are traditionally served under the `/static` path in your Django project. However if you would serve your `serviceworker.js` file on the `/static` path, the scope would also be limited to `/static`, and there is not a single HTML page here. Therefore, the easiest way to serve your `serviceworker.js` file is by using a Django view.

In this project, you can put this view in the polls app. It will then be served under `/polls/serviceworker.js` and its scope will be the polls apps custom views. In a real project you might want to serve your service worker in the namespace without URL prefix. It will then be served at `/serviceworker.js` and will have your whole project as its scope instead of just one app. Alternatively, you can also have several service workers, each with their own scopes. Scopes should not overlap though, as in that case only the most specific service worker will apply (and the newest one if they have exactly the same scope).

Add the following view to your polls app's `views.py`:

---

`polls/views.py`

...

```
class ServiceWorker(generic.TemplateView):
    template_name = "polls/serviceworker.js"
    content_type = "application/JavaScript"
```

Also add the the URL to the polls app's `urls.py`

---

`polls/urls.py`

...

```
urlpatterns = [
    ...
    url(r'^serviceworker.js$', views.ServiceWorker.as_view())
]
```

---

Finally, add the service worker itself. Create `polls/templates/polls/serviceworker.js` and add the following content to it:

---

```
polls/templates/polls/serviceworker.js
```

```
self.addEventListener("fetch", function(event){
  console.log("Fetch request for:", event.request.url);
});
```

---

This is a simple service worker that listens to fetch events, and logs them.

You now have created a service worker and Django is serving it at `/polls/serviceworker.js`. But you have not told the browser yet that you want to register the file at `/polls/serviceworker.js` as a service worker. This should be done from inside a JavaScript script in a page the user is visiting. So you will have to add another JavaScript file containing the service worker registration and include it in your templates. As this file is just a normal JavaScript file loaded directly in the page, and not a service worker you can add it to the static files.

Create a file called `app.js` in `polls/static/polls`, and add the following content to it:

---

```
polls/static/polls/app.js
```

```
if ("serviceWorker" in navigator){ // Check if browser supports service workers
  navigator.serviceWorker.register("/polls/serviceworker.js")
  //register returns a promise
  .then(function(registration){ // if promise resolved
    console.log("Service Worker registered with scope:", registration.scope);
  }).catch(function(err){ // if promise rejected
    console.log("Service worker registration failed:", err);
  });
}
```

---

Then add the following line to the scripts block in your `base.html` file:

---

```
polls/templates/polls/base.html
```

```
...
{% block scripts %}
  <script src="{% static 'polls/app.js' %}"></script>
{% endblock scripts %}
...
```

---

If you now run your Django development server (`python manage.py runserver`), and visit one of the pages from your polls app, for example `localhost:8000/polls/`, your service worker will be registered. You can check this by looking at the browser logs using the developer tools of your browser. When you subsequently click around within the polls app by opening questions and casting votes, you should see that each request is logged by your service worker.

**Note:** If you do not have any questions in your local database yet, use your superuser credential to go to the Django admin and create some.

## A.4.2 Letting Your Service Worker Serve an Offline HTML Page

Now that you have registered your service worker, in this subsection, you will use it to serve a customized offline page to your users. First, create an offline page in your polls app. To do this, you need to make the following changes:

Add a new file called `offline.html` in `polls/templates/polls/`:

---

`polls/templates/polls/offline.html`

---

```
{% extends "polls/base.html" %}

{% block title %} Polls {% endblock title %}

{% block content %}
<h1>You are offline</h1>
<p>
  We are sorry that you are experiencing connectivity issues
  and hope you come back soon to share your opinion with our polls.
</p>
{% endblock content %}
{% block scripts %}
{% endblock %}
```

---

Add the view to `polls/views.py`:

---

`polls/views.py`

---

```
...
class OfflineView(generic.TemplateView):
    template_name = "polls/offline.html"
```

---

And add the URL to `polls/urls.py`:

---

`polls/urls.py`

---

```
...
urlpatterns = [
    ...
    url(r'^offline/$', views.OfflineView.as_view(), name="offline")
]
```

---

Next, in order to be able to show the user this custom offline page that you have defined on your server, you will first have to store it in the users browser. A good time to do this is when installing your service worker. This way you can be sure that this page is available in the cache when your service worker becomes active after installation.

Remove the existing code from your `serviceworker.js` file and instead add the following:

---

```
polls/templates/polls/serviceworker.js
self.addEventListener("install", function(event){
  event.waitUntil(// dont' let install event finish until this succeeds
    caches.open("polls-cache")// open new or existing cache, returns a promise
      .then(function(cache){// if cache-opening promise resolves
        return cache.add("/polls/offline/")// add offline page
      })
    });
});
```

---

This adds an event listener for the install event. The install event happens after the registration and before the activation of your service worker. You will learn more about the service worker life cycle and which events happen when in the next section. You want the install event to succeed only when it manages to cache the offline page. If not the install event should fail, and reattempted later. This is achieved by using `event.waitUntil(function)`. The function passed to `waitUntil` should return a promise. Only if this promise resolves will the install event be allowed to end.

`caches.open` either opens an existing cache with the name you pass on to it, or, if no cache with this name exists, it creates a new one. The opened cache is returned in a promise. So after this promise resolves, we can be sure to have a usable cache instance, and add our offline page to it. Resources are stored in the cache with their path as a key. Old entries with the same key will be overwritten.

Now that you have put your offline page in the cache, you can retrieve it when fetching content from the server fails. To do this, add the following to your `serviceworker.js` file:

---

```
polls/templates/polls/serviceworker.js
self.addEventListener("fetch", function(event){
  event.respondWith(// answer the request with ...
    fetch(event.request) // try to fetch from server
      .catch(function(){ // if this is rejected
        return caches.match("/polls/offline/") // return offline page from cache
      })
    });
});
```

---

**Note:** The promise that is returned by `caches.match` always resolves, even if no match was found. In this case this does not really matter. We know we loaded the page during the install phase, and if for some reason it would not be there anymore, we would not be able to get it from the server anyway as we already know we are offline. In other situations you might want to check for it though:

---

```
    caches.match("request/path").then(function(response){
      if (response){
        return response;
      }
    });
```

---

Testing the new service worker is easiest when you delete the old one manually. In Firefox first go to the list of service workers by typing `about:debugging\#workers` in your address bar or by choosing `WebDeveloper>ServiceWorker` from the `tools` menu. In Chrome the list of service workers can be found under `chrome://serviceworker-internals`. Then click `unregister` next to your `localhost:8000/polls` service worker. Another way to make sure the service worker gets replaced is by closing all the tabs related to the polls app, and waiting until the service worker stops running. Only now, open a polls page in a new tab again for the register event to be triggered and installing the new service worker version. The following section on service worker life cycle will help you understand better as to why this is necessary.

Then run your Django development server (`python manage.py runserver`) and visit the page to let your new service worker install. In the shell where you are running the Django server, you should see that a get request for the `/polls/offline/page` is made. Then quit the development server (`ctrl-c`) and reload the page in your browser. You should now see your custom offline page.

### A.4.3 Intermezzo: Service Worker Life Cycle

What makes service workers slightly more complicated than normal JavaScript run in a webpage is the fact that they are not bound to a single page they can live and die with. They can control multiple pages in different browser windows or tabs, and can even react to certain events without needing any open tab or window at all. To handle this complexity correctly service workers have well defined life cycles with different states. You have already encountered the installation state, triggered by registering the service worker, and the active state, in which the service worker can listen to fetch events. But installed does not necessarily mean active. Before a newly installed service worker can become active, the browser needs to make sure that the old one is done with its work. So as long as there are pages controlled by the old service worker open, the new one will have to stay in the waiting state.

Figure A.1 shows a diagram of the service worker life cycle. The first state is the **installing** state. It is triggered by the register function, but only if the service worker that is asking to be registered is different from the currently active one. So you can be sure that one and the same service worker is only installed once, even if the register logic is hit multiple times. You can add your own logic

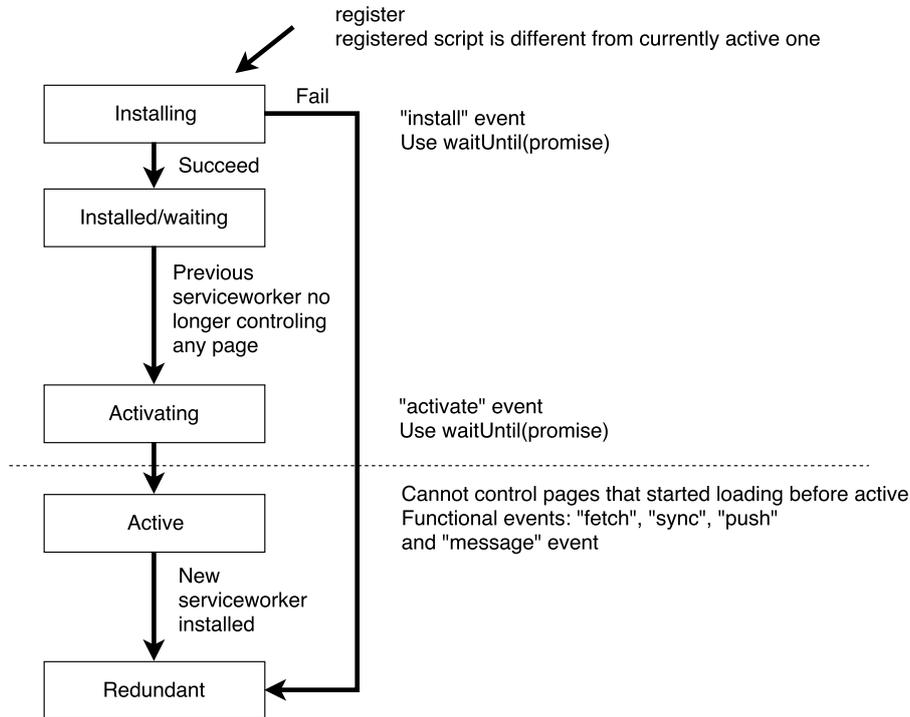


Figure A.1: Service worker's life cycle

to the installing phase by listening to the “install” event and using `waitUntil()` to keep the service worker in that state until your logic either succeeds or fails. If it fails, your service worker will go to the redundant state. Only when the `register` function is called again (for example by refreshing the page) will there be another install event and another chance to get your service worker active.

After successful installation, the service worker comes in a **waiting** state. If no active service worker is for the same scope is detected, it can move straight on to the next state. If there is an active service worker however, it has to wait here, until the active service worker is no longer controlling any pages and can be turned off and moved to the redundant state.

**Note:** Refreshing a page is not enough to make a service worker redundant and start using a new one. During refresh, the service worker will remain in control. The only way to stop it is to navigate to a page outside of its scope, or to close the tab it is controlling.

The service worker now enters the **activating** state. This is the place where you can do things that have to be done before the new service worker becomes active that might have a negative impact on an old service worker that might have been still running during the installing phase. To add such logic to your service worker listen to the “activate” event. Again you can use `waitUntil()` to avoid that the service worker moves on to the active state already and starts receiving fetch events.

After the activating process has finished, the service worker moves to the **active** state. In this state it can react to various events such as “fetch”, “sync”, “push” and “message”.

The final state of the service worker is **redundant**. Service workers will land in this state when registration (for example: the specified service worker path does not exist) or installation (the promise passed to the “install” events `waitUntil` method rejects) fail. Or when they are replaced by a new service worker.

#### A.4.4 Caching Multiple Assets and Cleaning up Old Caches

You might have noticed that the poll’s background image is not loaded as part of your offline page. That is because you have only cached the HTML page and not the additional assets such as CSS files and images. You will have to make to more changes to the `serviceworker.js` file to change this.

First change the install event logic like so:

---

```
polls/templates/polls/serviceworker.js
var CACHE_NAME = "polls-cache-v2";
var CACHED_URLS = [
  "/polls/offline/",
  "/static/polls/style.css",
  "/static/polls/images/background.png"
]

self.addEventListener("install", function(event){
  event.waitUntil(// dont' let install event finish until this succeeds
    caches.open(CACHE_NAME)// open new or existing cache, returns a promise
      .then(function(cache){// if cache-opening promise resolves
        return cache.addAll(CACHED_URLS)// add CACHED_URLS
      })
    );
});
...

```

---

Instead of `cache.add` the service worker now uses `cache.addAll` to add a list of URLs to the cache.

Second, you have to update the fetch event logic to return the cached assets, and not just the offline HTML page, as it is doing now. This can be done as follows:

---

```
polls/templates/polls/serviceworker.js
```

---

```
...
self.addEventListener("fetch", function(event){
  event.respondWith(// answer the request with ...
    fetch(event.request) // try to fetch from server
    .catch(function(){ // if this is rejected
      return caches.match(event.request) // try to return from cache
    })
    .then(function(response){ // caches.match always resolves
      if (response) { // check if something was found in cache
        return response;
      } else if
        (event.request.headers.get("accept").includes("text/html")){
        // requesting a html file and not finding a match in cache
        return caches.match("/polls/offline/")
      }
    })
  });
});
```

---

To test, remove the old service worker manually using the developer tools. Then run your Django development server (`python manage.py runserver`), and load a polls app page to register and install the new service worker. Quit the Django development server and reload the page in your browser. Does your offline page now have the background image?

Two questions remain to be answered. Why did I instruct you to change the cache name by adding a v2 tag to it? And, as a new cache was created instead of using the old one, what happens to the old caches?

There are a couple of reasons for versioning your cache. First, while the browser does detect changes to the `serviceworker.js` file automatically and reinstalls it, if we would make changes to any of the cached assets, this will not be seen by the browser. By changing the cache name each time you change something, you can make sure a new service worker is installed and your cached assets are updated. On top of that, the new assets will be downloaded in a new cache. This is good, because caching is done during the installation phase. This means the old service worker, that might rely on the old assets, is still active.

**Note:** While it is considered good practice to version your cache by changing the cache name with each version, you should try not to change your service worker URL as this can cause unwanted side effects. For example, suppose you have an active `serviceworker.js` that is active and caches rigorously. If you would change it to `serviceworker-v2.js`, you would then also have to change the argument of the register function, which might be part of the cached assets. How are you ever going to get this update live? It is better to keep the service worker URL the same, and let the browser take care of updating your service worker when something has changed.

You do, however, need to take care of deleting old caches. Browser storage is limited, and although the actual limits vary based on various circumstances, it is a good habit to clean up after yourself from the start. Cleaning up the cache is typically done in the activation phase of the service worker life cycle, as at this point you can be sure that the old service worker is no longer needing it.

Add the following code to your `serviceworker.js` file:

---

```
polls/templates/polls/serviceworker.js
```

---

```
self.addEventListener("activate", function(event){
  event.waitUntil( // do not finish activation until this succeeds
    caches.keys() // a promise containing an array of all cache names
      .then(function(cacheNames){
        return Promise.all(
          // return a promise that only succeeds
          // if all of the promises in the array succeed
          cacheNames.map(function(cacheName){
            // to create an array of promises, map the array of cachenames
            // for each of them:
            if (CACHE_NAME !== cacheName ){
              // if name is not the same as current caches name
              return caches.delete(cacheName);
              // delete it and succeeds after that
            }
          });
        });
      });
  });
});
```

---

This code listens to the “activate” event. It first gets an array of cache names. This array is then transformed to an array of promises, one for each cache we want to delete, succeeding exactly then when the cache is deleted. This array is passed to `Promise.all()`, which only succeeds if all elements of the array succeed.

**Note:** Depending on your application you might need a more complicated if clause. For example if you are using the cache storage at other places in your app, or if you use multiple caches at once.

You have now enhanced your Django web app with a custom offline page using a service worker. That is nice, but you can do more than that with service workers. In the next section you will implement a more complete caching strategy.

## A.5 Implementing a Caching Strategy

### A.5.1 Caching Patterns

Before starting implementing a caching strategy for the polls app, this subsection is dedicated to give an overview of some common caching patterns. Your caching strategy will typically include multiple of those patterns and variations on them, depending on the content.

**Cache only:** Assets that almost never change, such as certain CSS files and background or header images, can just be loaded once during the service worker installation, and then always be served from cache.

**Network only:** This is what would happen without a service worker. It can be achieved by simply ignoring the requests fetch event, which then by default gets past on to the server.

**Cache, fallback to network:** In this case, the service worker tries to find the asset in the cache first, and if this fails, it attempts to serve it from the network. This can be used for a type of assets that you know will never change once it is served, but for some reason you cannot or do not want to download it yet at the moment the service worker is installed.

**Network, fallback to cache:** Here you will always try to fetch the request from the network first, but if this fails, you will return the version from the cache.

**Cache then network:** This approach tries to be fast and up-to-date. It will first serve the version from the cache, as this is faster. Then it will try to fetch the same content from the network as well, and update it as soon as it arrives.

**Generic fallback:** This is similar to our custom offline page. When the network request fails, we will get some generic asset from the cache, which is better than just an error. It can also be used for images. If the service worker cannot get this specific image, it might return a generic image that has been cached during installation. This will look more in theme than just failing to load the image.

Although it is not mentioned explicitly, often when getting a response from the network, it is a good idea to create a new cache entry, or update the old one.

### A.5.2 Caching the Polls App

For this tutorial, I have chosen the following caching strategy:

- Static assets will be cached on service worker installation, and served from cache, falling back to network.
- For HTML pages the service worker will attempt to serve them from the network first, and cache them. If the network fails, it will try to serve them from the cache, and if they are not found in the cache, it will serve our custom offline page.

Here is what this looks like in code (note that the install and activate events remain the same, and are left out of the code snippet):

---

polls/templates/polls/serviceworker.js

```
var CACHE_NAME = "polls-cache-v3";
var CACHED_URLS = [
  "/polls/offline/",
  "/static/polls/style.css",
  "/static/polls/images/background.png",
  "/static/polls/app.js"
]

... // Install and activate events remain the same

self.addEventListener("fetch", function(event){
  var requestURL = new URL(event.request.url);
  // request for a html file
  if (event.request.headers.get("accept").includes("text/html")){
    event.respondWith( // answer the request with ...
      caches.open(CACHE_NAME).then(function(cache){ //open cache
        return fetch(event.request) // try to fetch from server
          .then(function(networkResponse){ //if succeeds
            if (networkResponse.ok){ //if it is not some errorpage
              //put a copy of the response in the cache
              cache.put(event.request, networkResponse.clone());
            };
            return networkResponse; //and return the response
          }).catch(function(){ // if fetching from server fails
            // return copy from cache or offline page from cache
            return cache.match(event.request).then(function(response){
              return response || cache.match("/polls/offline/");
            });
          });
      });
  });
});

//request is one of the resources cached during registration
} else if(
  CACHED_URLS.includes(requestURL.href) ||
  CACHED_URLS.includes(requestURL.pathname)
){
  event.respondWith( // answer the request with ...
    caches.open(CACHE_NAME).then(function(cache){ // open cache
      return cache.match(event.request).then(function(response){
        // return matched result from cache or try to fetch from network
        return response || fetch(event.request);
      });
    });
  });
};
});
```

---

The cache name is updated to `v3`. And the `app.js` file is now also added to the array of URLs to be cached during installation. The `install` and `activate` event logic remains the same. But the `fetch` event is updated to fulfill the new requirements. It exists of two parts, written as an `if` and an `else if` clause.

The `if` clause filters out all requests for HTML files. Then it opens the cache and tries to fetch those request from the network. If this succeeds, a response will be returned. A clone of this response has to be put in the cache, except if it was some kind of error response, as you do not want to clutter your cache with those. It is important to clone the response, as each response can only be consumed once. The response itself is returned to the page. If the network `fetch` fails, first the script tries to match the actual request with the cache entries. Remember that `caches.match()` succeeds even when nothing is found. Therefore the script returns this response or the offline page from the cache.

The second part (the `else if` clause) takes care of those assets that were cached during installation. It opens the cache and returns the response from it. This should normally be there, but if for some reason it isn't, we can still try to fetch it from the network.

Requests that do not fit any of those clauses will just pass trough the service worker and be requested from the server only.

Now, run your Django development server (`python manage.py runserver`) and test out your new service worker. Be sure not to forget to give your service worker the chance to be activated before you start testing its behavior.

### A.5.3 Letting the User Know the Content Might be Outdated

Thanks to our caching strategy users will be able to view some content even if they are offline. This also means, however, that they might see outdated content without noticing. Also, the way the polls app functions right now, users might try to submit a vote to a poll when offline, which won't work. From a usability point of view, we should let users know in advance, rather than letting them find out another way.

This looks more straightforward than it is. The service worker can fetch responses or craft its own, but it can not easily make changes to responses it got from the server or the cache. These are read-only. (If you want to try it anyway, Craig Russell wrote a blogpost about it [54].) From the page itself, we have easy access to the `navigator.onLine` property, however this is implemented differently by different browsers, and does not always mean the same as the service worker has served this page from cache. Because of this, the most recommended way to check whether or not someone is offline is by sending an AJAX request to the server.

When you have a kind of app-shell HTML page, that then uses an AJAX-request to get `data.json` from the server, this comes together really nicely.

You serve the shell HTML page from cache. It has some JavaScript code to (1) set up an eventlistener for messages from te service worker, and (2) request `data.json`. The service workers `fetch` event will probably contain a separate `if` clause for the `data.json` case. It might get the data with a network, fallback to cache approach with updating the cache. In the case the data comes from

the cache, the service worker can now additionally send a message to the page that the data has been served from the cache.

However this approach does not work when requesting the whole page at once, because by the time the eventlistener will be set up and running, the message has long been issued and gone.

For this tutorial, rather than issuing a useless AJAX request from the page, I chose to send a message to the service worker. The service worker reacts to this message by requesting a page from the server, and in case this fails, sends a message back to the page confirming the offline status. This approach is not perfect, as the page might have been delivered from the network if the network issues occurred right in between the page load and the message, but the result is usable and lets me demonstrate both directions of page - service worker messaging. There is more to messaging than this though. A service worker can also send broadcast messages to all of its pages, and you can set up a channel for communicating forth and back between a service worker and a page. Those things are not covered here.

First, set up the page side of the messaging. To do this, add the following code to the `app.js` file:

---

```
polls/static/polls/app.js
```

---

```
...
if ("serviceWorker" in navigator && navigator.serviceWorker.controller){
  // serviceWorker in navigator does not mean service worker installed.
  // we need controller to be available
  navigator.serviceWorker.addEventListener("message", function(event){
    // when receiving a message
    if (event.data === "offline"){ // if the data is "offline"
      // add an alert to the top of the page
      var newDiv = document.createElement('div')
      var newContent = document.createTextNode("You are currently offline. " +
        "The content of this page may be out of date.");
      newDiv.appendChild(newContent);
      newDiv.classList.add('alert')
      document.body.insertBefore(newDiv, document.body.firstChild);
      // if there is a submit button on the page, disable button
      var b = document.querySelector("input[type='submit']");
      if (b){
        b.setAttribute("disabled", "");
      }
    };
  });
  // send a message to the service worker
  navigator.serviceWorker.controller.postMessage("offline?")
}
```

---

If service workers are supported and a service worker is installed (controller is available), this code first sets up an event listener for message events. As there might at some point be different

messages we check for the message data, and if it fits we create a new div with a warning message. JQuery is not a dependency for this project, so vanilla JavaScript is used to do this. Of course you could also use JQuery if you want to. Add the alert class to the div so that you can style it later. Also, there is code to disable the submit button if there is one. This code disables the first submit button only, if you have more than one submit button, you would have to adapt it. After setting up the event listener, a message is sent to the service worker.

On the service worker side, we will update the version number and add a new event listener for messages like so (note that all other event listeners remain the same and are omitted in the code snippet):

---

polls/templates/polls/serviceworker.js

```
var CACHE_NAME = "polls-cache-v4";

... // CACHED_URLS and all other event listeners remain the same

self.addEventListener("message", function(event){
  if (event.data === "offline?"){ // check the message data
    fetch("/polls/").catch(function(response){
      // fetch from the network and if it fails
      self.clients.get(event.source.id).then(function(client){
        // get the client that sent this message
        client.postMessage("offline"); // and send a confirmation message
      });
    });
  }
});
```

---

While there is only one service worker, there can be multiple clients. The client-id of the client sending a message to the service worker is recorded in `event.source.id`. Based on this the service worker can get this client and post it a message. It does this only when the server is not reachable though. When the fetch request succeeds, no action is taken.

Finally, you can style the offline warning by adding some rules to your style.css file:

---

polls/static/polls/style.css

```
...

.alert{
  background-color: #F8D7DA;
  color: #7A1C2F;
}
```

---

Time to test! Fire up your Django development server (`python manage.py runserver`), let your service worker install and activate, and browse some pages to cache them. Then turn off the server, and revisit those pages. You should now see the warning message and the voting button should be disabled.

**Note:** Django serves static files with a longer cache header. This becomes very annoying when working with service workers. There is no such thing as reload service worker while ignoring cache, as there is for reload page. So you might have to clear your browser cache manually for this to work. This is a good reminder that you will have to take care of what cache header your static files will be served with in production.

## A.6 What is Next?

You now have a pretty good idea of what a service worker is and how you can use it for caching specific pages of your website. You also got acquainted with the service worker lifecycle and might have encountered some service worker pitfalls already.

The one thing holding me back from using service workers for more complex things is that there is still no easy solution for automated testing of service workers. Lack of browser support for the newer APIs could be another minor reason, but this one can be counted for by progressive implementation and will solve itself eventually.

I learned a lot from the book “Building progressive web apps” by Tal Ater [2]. If you want to look online for resources, you might want to search for ‘indexedDB’, ‘Background Sync’ and ‘Push Notifications’. Also not covered in this tutorial is the Web app manifest.

## Appendix B

# Static Example Infographics Using HTML and Bootstrap

### Travel

Hi,  
I am Sara.  
I have autism.



This situation is very stressful to me

- I might not be able to talk or answer your questions
- I might be able to write if you give me pen and paper or type on my smartphone
- Please don't touch me or try to comfort me by talking a lot
- **Please help me get back to my hotel**

### Contacts



**Hotel Imperial**  
Gran Via de les Corts  
Catalanes, 668  
08010 Barcelona,  
Spanien  
Tel: +34 935 10 11 30

**Emergency contact**  
John Doe  
Tel: +41 76 123 45 67

Figure B.1: Static HTML example infographic for traveling

## Hairdresser visit

**Hi,**  
I am Sara.  
I am autistic.



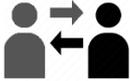
Read on to learn more about what autism means to me and how you can help me.

### Communication

**In general**

You can use

- Spoken language
- Written language



I will use

- Spoken language
- Written language

**When I am stressed**

You can use

- Spoken language
- Written language



I will use

- Written language

### Do's and Don'ts

- Don't ask me too many questions
- Don't make unexpected hard noises such as slamming the door

- Ask before touching my stuff
- Ask me before using products with a pronounced smell

- Leave me enough time to answer your questions or to make decisions
- Tell me what is going to happen beforehand

### This is what I want my hair to look like



As my hair is curly it will be shorter when dry. Please don't cut it too short as I still want to be able to bind it together in a ponytail.

Figure B.2: Static HTML example infographic for hairdressers visit

## Doctors visit

**Hi,**  
I am Sara.  
I have an Autism Spectrum Condition.



Read on to learn more about what autism means to me and how you can help me.

### Do's and Don'ts

- Don't touch me without permission
- Don't ask me too many questions

- Ask before touching my stuff
- Ask me if I am ready to go before you take me to a new place

- Give me clear instructions if I have to do something
- Leave me enough time to answer your questions or to make decisions
- Tell me what is going to happen beforehand
- Write important information down for me.

### Sensory sensitivities



I am very sensitive to sound



I am very sensitive to light  
I can't cope with fluorescent lighting



I am less sensitive than average to temperature



I have trouble processing more than one sense at a time, for example hearing you while looking at something  
I have difficulties recognizing and/or reporting pain or other symptoms

### Medication

| What         | When            | Quantity | Remarks                             |
|--------------|-----------------|----------|-------------------------------------|
| Wonder pills | Breakfast       | 2        | Take with my favorite banana yogurt |
|              | Dinner          | 1        |                                     |
| sweet sirup  | before sleeping | 200 ml   |                                     |

### Contacts

**My GP (General Practitioner)**  
Dr. Jef Arzt  
Praxisstrasse 1  
3000 Bern  
Tel: 031 123 45 67

Figure B.3: Static HTML example infographic for doctors visit

# List of Tables

|     |   |    |
|-----|---|----|
| 5.1 | Iterative development and testing, an overview of versions and test persons . . . . | 26 |
|-----|---|----|

# List of Figures

|     |   |    |
|-----|---|----|
| 1.1 | Examples of autism id cards . . . . .   | 4  |
| 3.1 | Screenshot of the AASPIRE healthcare tool . . . . .   | 11 |
| 3.2 | Paper prototype of the “General info module” . . . . .  | 13 |
| 3.3 | Paper prototype of the “Communication module” . . . . .   | 13 |
| 3.4 | Paper prototype of the “Do’s and Don’ts module” . . . . .   | 13 |
| 3.5 | Paper prototype of the “Communication module” form . . . . .  | 14 |
| 4.1 | Graphical representation of models . . . . .  | 19 |
| 4.2 | Overview of the registration and login process . . . . .  | 23 |
| 5.1 | The navbar in Version 0.4 with the call to action button and items that are dead links, and the improved one in Version 0.4.1 . . . . .   | 27 |
| 5.2 | Additional information on the homepage about the purpose of the webapp. . . . .   | 28 |
| 5.3 | While creating the infographic users see a list of modules they have already created. The Preview button on the top right (circled in red) was added in Version 0.4.1 . . . . . | 29 |
| 5.4 | List of modules and submit button: Version 0.4 is shown on the left side. The frame on the right shows the improvement made in Version 0.4.1. . . . .                           | 30 |
| 5.5 | Top: The message at the top of the form in Version 0.4.1 was easily overlooked by users. Bottom: Improved message in Version 0.5. . . . .                                       | 32 |
| A.1 | Service worker’s life cycle . . . . .   | 46 |
| B.1 | Static HTML example infographic for traveling . . . . .   | 58 |
| B.2 | Static HTML example infographic for hairdressers visit . . . . .  | 59 |
| B.3 | Static HTML example infographic for doctors visit . . . . .   | 60 |

# Bibliography

- [1] American Psychiatric Association et al. *Diagnostic and statistical manual of mental disorders (DSM-5®)*. American Psychiatric Pub, 2013.
- [2] Tal Ater. *Building progressive web apps: bringing the power of native to the browser.* ” O’Reilly Media, Inc.”, 2017.
- [3] Autisticallity. *Neurodiversity, language, and the social model*. URL: <https://autisticallity.com/2014/10/27/neurodiversity-language-and-the-social-model/> (visited on 17/03/2018).
- [4] Simon Baron-Cohen. “Is Asperger syndrome/high-functioning autism necessarily a disability?” In: *Development and psychopathology* 12.3 (2000), pp. 489–500.
- [5] Luke Beardon. *How can unhappy autistic children be supported to become happy autistic adults?* Presentation. 2017. URL: <https://blogs.shu.ac.uk/autism/files/2017/07/How-can-unhappy-autistic-children-be-supported.pptx> (visited on 17/03/2018).
- [6] Sander Begeer et al. “Underdiagnosis and referral bias of autism in ethnic minorities”. In: *Journal of autism and developmental disorders* 39.1 (2009), p. 142.
- [7] Penny Benford and P Standen. “The internet: a comfortable communication medium for people with Asperger syndrome (AS) and high functioning autism (HFA)?” In: *Journal of Assistive Technologies* 3.2 (2009), pp. 44–53.
- [8] Laura Benton et al. “Developing IDEAS: Supporting children with autism within a participatory design team”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2012, pp. 2599–2608.
- [9] Laura Benton et al. “Diversity for design: a framework for involving neurodiverse children in the technology design process”. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM. 2014, pp. 3747–3756.
- [10] Laura Benton et al. “IDEAS: an interface design experience for the autistic spectrum”. In: *CHI’11 Extended Abstracts on Human Factors in Computing Systems*. ACM. 2011, pp. 1759–1764.
- [11] Hanna Bertilsdotter Rosqvist, Charlotte Brownlow, and Lindsay O’Dell. “Mapping the social geographies of autism—online and off-line narratives of neuro-shared and separate spaces”. In: *Disability & Society* 28.3 (2013), pp. 367–379.
- [12] Bootstrap. *Bootstrap Documentation: Introduction*. URL: <https://getbootstrap.com/docs/4.0/getting-started/introduction/> (visited on 17/03/2018).

- [13] Priscilla Braz et al. “An alternative design perspective for technology supporting youngsters with autism”. In: *International Conference on Augmented Cognition*. Springer. 2014, pp. 279–287.
- [14] Kelly D Coons and Shelley L Watson. “Conducting Research with Individuals Who Have Intellectual Disabilities: Ethical and Practical Implications for Qualitative Research.” In: *Journal on Developmental Disabilities* 19.2 (2013).
- [15] Joyce Davidson. “Autistic culture online: virtual communication and cultural expression on the spectrum”. In: *Social & cultural geography* 9.7 (2008), pp. 791–806.
- [16] Martijn Dekker. “On our own terms: emerging autistic culture”. In: *Conferencia en linea*. 1999.
- [17] MA Dziuk et al. “Dyspraxia in autism: association with motor, social, and communicative deficits”. In: *Developmental Medicine & Child Neurology* 49.10 (2007), pp. 734–739.
- [18] Jessica Enders. *Designing UX: Forms: Create Forms That Don't Drive Your Users Crazy*. SitePoint, 2016.
- [19] Marc Fabri and Penny CS Andrews. “Human-centered design with autistic university students: interface, interaction and information preferences”. In: *International Conference of Design, User Experience, and Usability*. Springer. 2016, pp. 157–166.
- [20] Marc Fabri, Penny CS Andrews, and Heta K Pukki. “Using design thinking to engage autistic students in participatory design of an online toolkit to help with transition into higher education”. In: *Journal of Assistive Technologies* 10.2 (2016), pp. 102–114.
- [21] Farhadur Reja Fahim. *Django registration with confirmation email*. URL: <https://medium.com/@frfahim/django-registration-with-confirmation-email-bb5da011e4ef> (visited on 17/03/2018).
- [22] Andrew Fenton and Tim Krahn. “Autism, neurodiversity and equality beyond the’normal’”. In: (2009).
- [23] Mignon Fogarty. *Gender-Neutral Pronouns: Singular They*. URL: <https://www.quickanddirtytips.com/education/grammar/gender-neutral-pronouns-singular-they> (visited on 17/03/2018).
- [24] The Django Software Foundation. *Django: The web framework for perfectionists with deadlines*. URL: <https://www.djangoproject.com/> (visited on 17/03/2018).
- [25] The Django Software Foundation. *Django: The web framework for perfectionists with deadlines*. URL: <https://docs.djangoproject.com/en/1.11/> (visited on 17/03/2018).
- [26] Vitor Freitas. *How to create a one time link*. URL: <https://simpleisbetterthancomplex.com/tutorial/2016/08/24/how-to-create-one-time-link.html> (visited on 17/03/2018).
- [27] Rebecca Grzadzinski, Marisela Huerta, and Catherine Lord. “DSM-5 and autism spectrum disorders (ASDs): an opportunity for identifying ASD subtypes”. In: *Molecular autism* 4.1 (2013), p. 12.
- [28] New York Times Harvey Blume. *Autistics Are Communicating in Cyberspace*. June 1997. URL: <https://partners.nytimes.com/library/cyber/techcol/063097techcol.html> (visited on 17/03/2018).
- [29] Tom Hutchison. “The classification of disability.” In: *Archives of disease in childhood* 73.2 (1995), p. 91.

- [30] Jacob Kaplan-Moss. *Django gotcha: concrete inheritance*. URL: <https://jacobian.org/writing/concrete-inheritance/> (visited on 17/03/2018).
- [31] Steven Kapp et al. “Deficit, Difference, or Both? Autism and Neurodiversity”. In: 49 (Apr. 2012).
- [32] Lorcan Kenny et al. “Which terms should be used to describe autism? Perspectives from the UK autism community”. In: *Autism* 20.4 (2016), pp. 442–462.
- [33] Jamie Knigh. *Cognitive Accessibility 101 - Part 2: How it effects me & the tools I use*. URL: <http://jkg3.com/Journal/cognitive-accessibility-101-part-2-how-it-effects-me-the-tools-i-use> (visited on 17/03/2018).
- [34] Meng-Chuan Lai et al. “Sex/gender differences and autism: setting the scene for future research”. In: *Journal of the American Academy of Child & Adolescent Psychiatry* 54.1 (2015), pp. 11–24.
- [35] Raphael Merx. *Offline Django with Service Workers (video)*. URL: <https://www.youtube.com/watch?v=70L8saIq3uo> (visited on 17/03/2018).
- [36] Raphael Michel. *Data internationalization in Django (video)*. URL: <https://www.youtube.com/watch?v=vdbpin0mMeo> (visited on 17/03/2018).
- [37] Laura Millen, Rob Edlin-White, and Sue Cobb. “The development of educational collaborative virtual environments for children with autism”. In: *Proceedings of the 5th Cambridge Workshop on Universal Access and Assistive Technology, Cambridge*. Vol. 1. 2010, p. 7.
- [38] Jodi Murphy. *Having a Meltdown? Use the Emergency Chat App for Help*. Aug. 2015. URL: <https://geekclubbooks.com/2015/08/meltdown-emergency-chat-app-help/> (visited on 17/03/2018).
- [39] Autism Self Advocacy Network. *About Autism*. URL: <https://autisticadvocacy.org/about-asan/about-autism/> (visited on 17/03/2018).
- [40] Christina Nicolaidis et al. ““Respect the way I need to communicate with you”: Healthcare experiences of adults on the autism spectrum”. In: *Autism* 19.7 (2015), pp. 824–831.
- [41] Christina Nicolaidis et al. “The development and evaluation of an online healthcare toolkit for autistic adults and their primary care providers”. In: *Journal of general internal medicine* 31.10 (2016), pp. 1180–1189.
- [42] Donald A Norman and Stephen W Draper. “User centered system design”. In: *Hillsdale, NJ* (1986), pp. 1–2.
- [43] Gregory O’Brien and Joanne Pearson. “Autism and learning disability”. In: *Autism* 8.2 (2004), pp. 125–140.
- [44] World Health Organisation. *Disabilities*. URL: <http://www.who.int/topics/disabilities/en/> (visited on 17/03/2018).
- [45] World Health Organization. *International statistical classification of diseases and related health problems, tenth revision*. Vol. 1. World Health Organization, 2004.
- [46] Michael Palmer and David Harley. “Models and measurement in disability: an international review”. In: *Health Policy and Planning* 27.5 (2012), pp. 357–364.
- [47] Nikolay Pavlov. “User interface for people with autism spectrum disorders”. In: *Journal of Software Engineering and Applications* 7.02 (2014), p. 128.

- [48] H Plattner. *An Introduction to Design Thinking Process Guide. The Institute of Design at Stanford*. 2010.
- [49] PRWEB. *Autistic Teens Create Website for People with Asperger’s Syndrome*. July 2004. URL: <http://www.prweb.com/releases/2004/07/prweb137362.htm> (visited on 17/03/2018).
- [50] Arun Ravindran. *Django Design Patterns and Best Practices*. Packt Publishing Ltd, 2015.
- [51] Academic Autism Spectrum Partnership in Research and Education. *AASPIRE: About*. URL: <https://aaspire.org/about/> (visited on 17/03/2018).
- [52] Academic Autism Spectrum Partnership in Research and Education. *AASPIRE: CBPR*. URL: <https://aaspire.org/about/cbpr/> (visited on 17/03/2018).
- [53] Academic Autism Spectrum Partnership in Research and Education. *AASPIRE Healthcare Toolkit: Patient Centered Care Tools for Autistic Adults*. URL: <https://autismandhealth.org/> (visited on 17/03/2018).
- [54] Craig Russell. *Modifying responses in Service Worker requests*. URL: <http://craig-russell.co.uk/2016/01/26/modifying-service-worker-responses.html> (visited on 17/03/2018).
- [55] Noah J Sasson et al. “Neurotypical peers are less willing to interact with those with autism based on thin slice judgments”. In: *Scientific reports* 7 (2017), p. 40700.
- [56] Douglas Schuler and Aki Namioka. *Participatory design: Principles and practices*. CRC Press, 1993.
- [57] Stacy Shumway et al. “The ADOS calibrated severity score: relationship to phenotypic variables and stability over time”. In: *Autism Research* 5.4 (2012), pp. 267–276.
- [58] Jim Sinclair. *Autism network international: the development of a community and its culture*. Mar. 2005. URL: [http://autreat.com/History\\_of\\_ANI.html](http://autreat.com/History_of_ANI.html) (visited on 17/03/2018).
- [59] Jim Sinclair. “Don’t mourn for us. Our voice”. In: *Autism Network International* 1.3 (1993). URL: [http://www.autreat.com/dont\\_mourn.html](http://www.autreat.com/dont_mourn.html) (visited on 17/03/2018).
- [60] Jim Sinclair. “Why I dislike ”person first” language”. In: *Autonomy, the Critical Journal of Interdisciplinary Autism Studies* 1.2 (2013).
- [61] J Singer. “Odd people in: The birth of community amongst people on the autistic spectrum: a personal exploration of a new social movement based on neurological diversity”. MA thesis. University of Technology, Sydney, 1998.
- [62] Judy Singer. “Why can’t you be normal for once in your life? From a problem with no name to the emergence of a new category of difference”. In: *Disability discourse* (1999), pp. 59–70.
- [63] National Autistic Society. *Designing autism friendly websites*. URL: <http://www.autism.org.uk/professionals/others/website-design.aspx> (visited on 17/03/2018).
- [64] W3C. *Web Content Accessibility Guidelines (WCAG) 2.1*. URL: <https://www.w3.org/TR/WCAG/> (visited on 17/03/2018).
- [65] Nick Walker. *Neurodiversity: Some Basic Terms & Definitions*. URL: <http://neurocosmopolitanism.com/neurodiversity-some-basic-terms-definitions/> (visited on 17/03/2018).
- [66] Amy S Weitlauf et al. “Brief report: DSM-5 levels of support: A comment on discrepant conceptualizations of severity in ASD”. In: *Journal of autism and developmental disorders* 44.2 (2014), pp. 471–476.