# Software Design Document

DIGITALIZATION MADE ACCESSIBLE

Names:

Shaul Taragin - 209337161

Ido Bar - 207765652

Noam Vanunu -318995156

Yuval Sandler - 319097036

Date: (12/01/2023)

## TABLE OF CONTENTS

# 1. INTRODUCTION

## 1.1 Purpose

This software design document describes the architecture and system design of **D**igitalization **M**ade **A**ccessible (or DMA) System.

## 1.2 Scope

Our web-based app aims to assist people in navigating the transition from traditional services to digital platforms, using built-in guides that show step-by-step user instructions.

The scope of the software includes providing tools for individuals to easily access and use digital services, as well as offering guidance for those who may have difficulty with technology. The app will have a user-friendly interface and interactive guides that will help users complete digital platforms tasks easily.

Our goals are to provide support for individuals who may have difficulty with technology and Make the transition to digital platforms more accessible and user-friendly.
The objectives of the project are to develop a user-friendly interface for accessing digital services and to create interactive step-by-step guides for digital tasks.
The benefits of the project are improved digital literacy, leading greater participation in the digital economy, and assisting individuals in navigating the transition to digital platforms for traditional services.

## 1.3 Overview

This article provides an overview of the core elements of our project.
**In section 2**, we give some background information for our project, discuss its relevance in today's world,and briefly go over some of its features.
**In section 3**, we present our architectural design, we describe the components that are part of our project, their purpose, and the technologies used to implement them.
**In section 4**, we elaborate on the data entities that exist in our projects,the data structure used to represent them,how we save them and where.
**In Section 5**, we delve into our components main features, and lay out the pseudocode of the functions that are tasked with implementing them.
**In section 6**, we talk about the visible features of our UI and how the user may use them,
as well providing demo screenshots of the UI.
Later in that section, we provide diagrams that show the flow that happens behind the scenes when a user interacts with the UI.
**In Section 7**, we refer the project's SRS and provide a table that show witch of our system components is in charge of this feature

# 2. SYSTEM OVERVIEW

Our project is designed to make digitalization more accessible. We aim to assist individuals who may not be as familiar with technology in navigating digital platforms.

The way we chose the achieve this goal, is to build a browser extension that will help users to juse digital services.
Some of our functional features include: Detecting the user's URL and having a pop-up offering him the guides for the website he is on, guiding the user by highlighting and marking the next move he shall take, and more.
The design of the extension is user-friendly and easy to navigate, with clear and concise instructions.
The project is relevant in today's digital age where many services have moved online, but not everyone may have the same level of tech literacy. This extension will help bridge that gap and make digitalization more accessible to all.

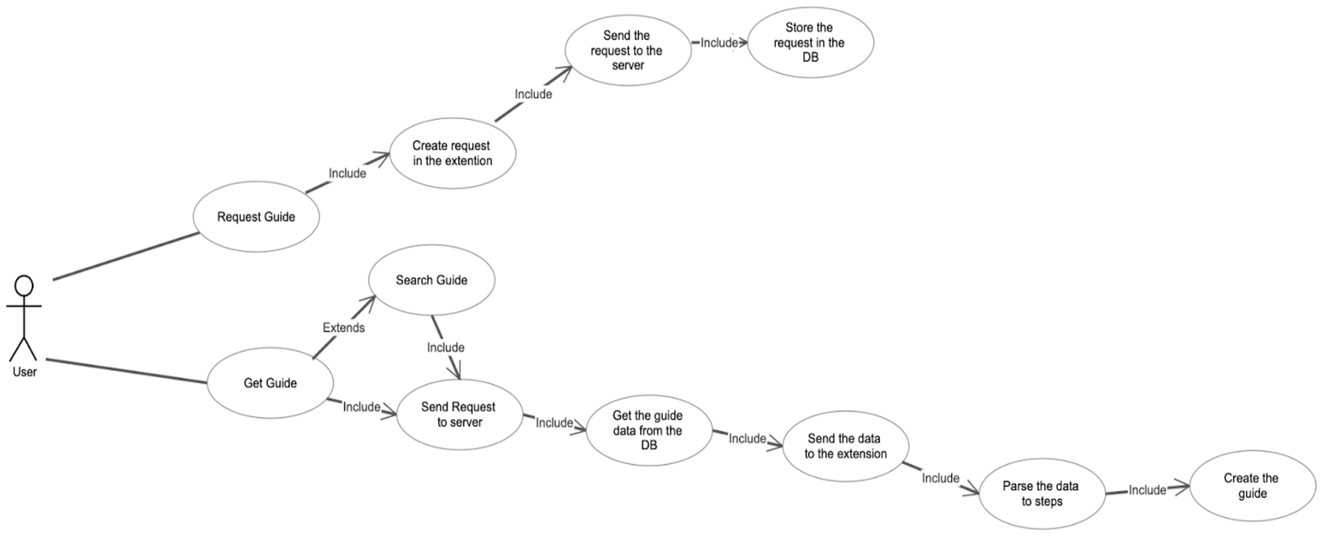
# 3. SYSTEM ARCHITECTURE

## 3.1 Architectural Design

Our system divided into 3 subsystems:

1. **Browser Extension** - An interactive browser extension that monitors URL addresses and suggests relevant guides based on the address visited. Additionally, the extension allows users to request a guide for forms that do not have one currently available.
   **Implementation:** HTML, CSS, JS - React.
2. **DataBase** - A database that will maintain a chronological order of the necessary actions required to fully complete a form. This will include a step-by-step guide for filling out the form.
   **Implementation:** mongoDb.
3. **Server:** A server that communicates with the database and processes the received data. The server will transmit the data to the browser extension that will generate a template for filling out the form. This will ensure that users have the necessary information and guidance to properly and efficiently complete forms.
   **Implementation:** NodeJs.

## 3.2 Decomposition Description



## 3.3 Design Rationale

We chose those three components in order to create a robust and efficient solution for our problem.

1. **Browser Extension**: The browser extension is the primary user-facing component, it monitors the URLs visited by the user and suggests relevant guides, this allows the user to quickly access the guides they need. Additionally, it allows the user to request a guide for forms that do not have one currently available, this feature allows users to request new guides and makes the system more user-friendly and improves the guide database.
2. **Database**: The database stores all the forms and their corresponding guides in a chronological order, this allows to save the form without actually saving the form itself.
3. **Server**: The server communicates with the database and processes the received data, it then transmits the data to the browser extension to generate a template for filling out the form. This ensures that the users have the necessary information and guidance to properly and efficiently complete forms.

# 4.Data description and dictionary

Since our project is a web service that doesn't need to track its users, and we don’t want to keep user's data (which might be sensitive considering our project nature), we don’t have too many data and system entities, however we still got a few of them.

- Databases and storage items used

    1. Mongodb – used to save the guide instructions

    2. Javascript basic data structures (lists , hash maps etc…)

- Major entities storage methods

    1. Guide instruction list – a list of key pair values each describing an instruction for the GuideRunner to execute. These instruction lists will be saved in MongoDb and will be fetched using the NodeJs server when a user will ask for a specific guide.

    2. Supported website – key value object, where the key is a url of a site that we have a guide for, and the value is the aforementioned list that describes the available guide for the site.

    3. Guide request – a key pair vale where the key is a url of site, and the value a string describing the requested action the client want us to include

# 5. COMPONENT DESIGN

Extension functions -

- Request Guide - pseudocode
    - Extract url from the requested site.
    - Prompt the user to enter a descriptive message of the requested guide.
    - get the description from the user, and create a pair of url-descriptions.
    - Send the pair to the server which will forward it to the database for future inspection.

- Execute Guide -
    - Get the requested Guide id from user input (click).
    - Send a request to the server to get the instructions list for the chosen guide.
    - Execute the instruction list using the GuideRunner.

Server functions -

- Fetch guide -
    - Receive id of Guide from extension.
    - Query MongoDb and get its instruction list.
    - return the instruction list to the caller.

- Post request
    - Get GuideRequest from the extension.
    - send to to MongoDb.

# 6. HUMAN INTERFACE DESIGN

## 6.1 Overview of User Interface

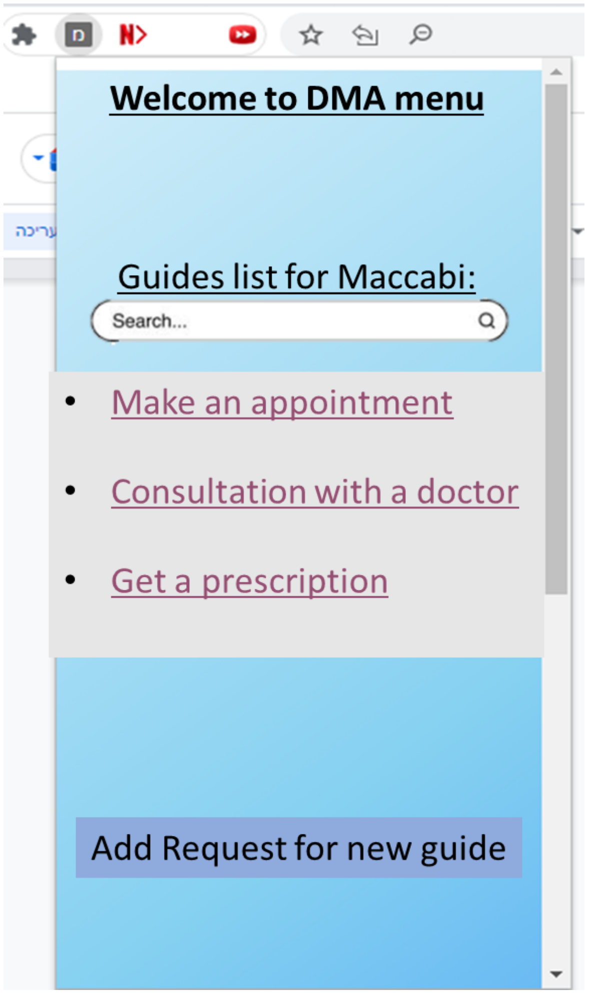
Scenario 1: Request for a non-exist guide- the client chooses the new guide option in the browser extension menu, and fills in the new guide details. The details are saved in the database and will be used later to create the requested guide.

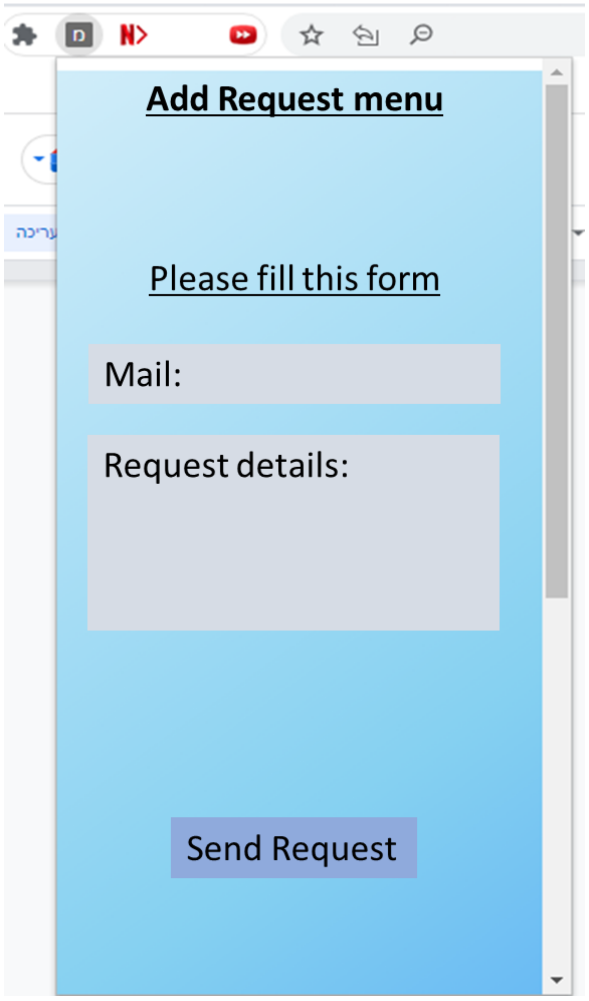
Scenario 2: Using an existing guide- the client chooses the guide option in the browser extension menu and chooses a specific guide from the guides list (that shows only the relevant guides for his current URL).
After confirming his choice, the interactive GuideRunner will execute the guide's instruction list.

Scenario 3: Filters the guides list- the client chooses the guide option in the browser extension menu, filters the guides list by the relevant features, and chooses a specific guide from the filtered guides list.
This will help the user navigate quickly in big sites that potentially could contain dozens of guides.

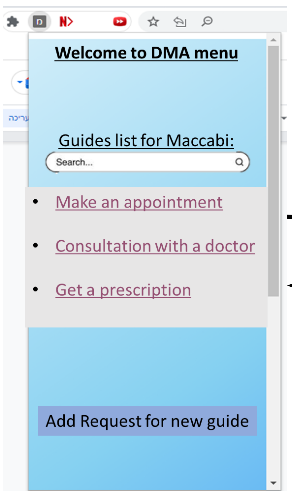
## 6.2 Screen Images

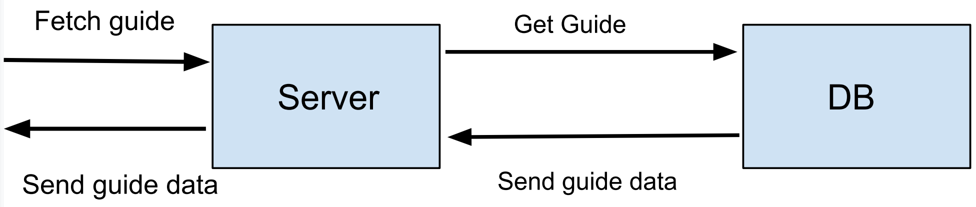
These are Demo screenshots (without any design yet), of the extension UI.

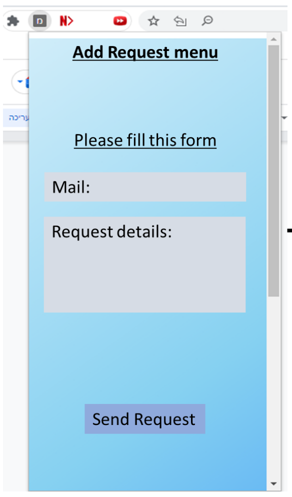


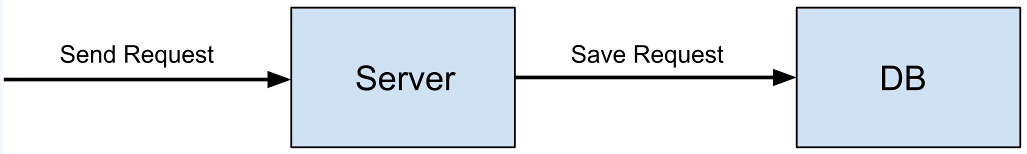


## 6.3 Screen Objects and Actions









# 7. REQUIREMENTS MATRIX

Some of the features from the SRS, and the components that taking care of them

| SRS ID | Requirement | Which component meets the requirement |
|---|---|---|
| 2. | Identifying the URL of the user. | Wrapper class that uses the browser extension API |
| 3. | Implementing an interactive guide which instructs the user in the website. | craft manually and stored in MongoDB database |
| 5. | Presenting to the user the relevant guides according to the URL. | Extension UI+MongoDB |
| 7. | Allowing users to request a guide that does not exist yet. | Extension UI |
| 8. | Highlighting and marking HTML elements that are part of the manual. | Extension guide runner |
| 13. | Option for user to report a malfunctioning guide | Extension UI |