# CSCB024 Практика по програмиране и интернет технологии

# Text Grouping Survey Website

## F64253 Deia Georgieva

## F97829 Tsvetelina Ivanova

## F95830 Samuela Nenova

## F102908 Konstantin Terziev

## Table of Contents:

## Task Goal

## Implementation Overview

## Database

## Backend

## Frontend

## Scenarios Covered

## Running the Project on a Local Machine

## Hosting

## Team tasks

# Task Goal

The goal of this project is to create a website for an Internet survey that involves grouping eight texts at a time into several groups by the emotion invoked in the participants. The texts can be grouped into no more than five groups and at minimum one, no limitation how many texts can be grouped together or how many times the texts can be moved around until the final submission.

The requirements for the number of texts and the number of groups have been developed with the assistance of a specialist in psychology statistical methods. The collected data will be processed by a special methodology, allowing finding distances and compiling a space with axes.

The texts were first selected by English specialists and then distributed in different variants - eight texts in each variant, out of a total of twenty texts included in the study. The task in making the variants is for each of the twenty texts to appear approximately the same number of times in the variants. There are over 400 different variants that can be used in the survey all kept in a separate table in the database.

# Implementation Overview

The task has been implemented using MySQL as the database provider, C# with .NET Core for the backend and Javascript with React for the frontend.

Libraries and other dependencies:

|  |  |
| --- | --- |
| Backend Libraries | Version |
| Microsoft.AspNetCore.SpaServices.Extensions | 3.1.10 |
| Microsoft.EntityFrameworkCore | 3.1.10 |
| Microsoft.EntityFrameworkCore.Design | 3.1.10 |
| Microsoft.EntityFrameworkCore.Tools | 3.1.10 |
| MySql.Data.EntityFrameworkCore | 8.0.22 |
| Microsoft.VisualStudio.Azure.Containers.Tools.Targets | 1.10.9 |
| Microsoft.VisualStudio.Web.CodeGeneration.Design | 3.1.4 |

|  |  |
| --- | --- |
| Frontend Libraries | Version |
| @atlaskit/theme | 11.0.2 |
| @emotion/react | 11.1.4 |
| @emotion/styled | 11.0.0 |
| @material-ui/core | 4.11.2 |
| @material-ui/lab | 4.0.0-alpha.57 |
| axios | 0.21.0 |
| bootstrap | 4.1.3 |
| immutability-helper | 3.1.1 |
| jquery | 3.4.1 |
| merge | 1.2.1 |
| oidc-client | 1.9.0 |
| react | 16.14.0 |
| react-beautiful-dnd | 13.0.0 |
| react-dom | 16.14.0 |
| react-router-bootstrap | 0.25.0 |
| react-router-dom | 5.1.2 |
| react-scripts | 3.4.1 |
| reactstrap | 8.4.1 |
| rimraf | 2.6.2 |
| styled-components | 5.2.1 |

# Database

The database is a MySQL database that consists of several tables:

* User – This table holds information about registered users, with the user email as the primary key.
* Survey – This table holds information about any surveys the users have started.
* GroupTextMapping – This table holds information about each individual action in a survey, e.g. a user dragging a specific text into a specific group creates a new row for this action. A user dragging a text away from a group and into another will cause the row to be marked as deleted and new row with the new mapping will be created. If the user drags the text away from a group back into the starting area, the row is just marked as deleted, but a new one will not be created at this time.
* TextEntry – This table holds all currently available texts and their assigned ids.
* Variant – This table holds all variants, that are combinations of the texts to be shown to the user. Each survey has its own variant, and a user can receive each variant only once.
* Group – This table holds the ids for the different groups that a text can be assigned to.
* Password – This table currently has only a single row, that holds the password hash and password salt for the password for the administrator panel.

# Backend

The backend is a standard ASP .NET Core Web API, that consists of three layers, data, service and web. The data layer uses Entity Framework Core as a ORM that translates C# code into SQL statements, with C# classes that correspond to each database table. The approach used is Code-First, which means that for every schema update required, Entity framework will generate a new database migration that can be run directly on the database.

The Service layer abstracts the database access and business logic away from the controllers in the Web layer. For each type of data, there is a separate service class, e.g. there is a SurveyService that creates and updates surveys when necessary and is called by the SurveyController class in the Web layer.

The Web layer contains the WebAPI controllers for each type of data. They can handle GET, POST and PUT requests, to create, update and retrieve data.

# Frontend

The frontend is a single-page application, written in Javascript, using React as the main library. The unique challenge of the drap-and-drop functionality in the main part of the survey has been resolved using Atlassian’s library react-beautiful-dnd. The client app is hosted within the .NET WebAPI web project under the ClientApp folder in the SurveyApp.Web project. The files are divided into separate component folders for each page, e.g. in the “components” folder there are admin, registration and survey folders that contain the multiple files used for each page. There are also common components, Header and Footer that are present on each page.

# Scenarios Covered

* New user registers and starts new survey.
* Existing user logs in with email to continue survey.
* Existing user can start new survey variant.
* Administrator logs in to access User, Survey and Mapping tables, and to download matrix.csv

# Running the Project on a Local Machine

In order to run the project locally, the following are required:

Visual Studio 2019 (optional, but necessary if you need to view the code)

NET Core SDK 3.1

Node.js (doesn’t matter what version)

Steps after installing the prerequisites:

1. Either open Visual Studio or navigate to the project directory in a terminal.
2. Build the project in VS, or run “dotnet build” in the project directory.
3. Ctrl + f5 will start a local server and open the UI in browser (from Visual Studio)
4. In the terminal the way to run it is “dotnet run”, which will show you what port it is currently running on and you can navigate to it in the browser.

# Hosting

The project is currently hosted on Ubuntu 18.04 OS using the following setup:

Web server: nginx 1.18.0-0ubuntu1 with reverse proxy to the kestrel server that the .NET Core app runs on

Certificate management: certbot 0.40.0-1ubuntu0.1 that has registered the certificate with Let’s Encrypt and will auto-renew the certificate before it expires (More information here: <https://certbot.eff.org/lets-encrypt/ubuntubionic-nginx.html>)

Domain: The domain has been acquired from <https://www.freenom.com> and is free for the first year. The DNS has been setup through the Digital Ocean UI, and can be managed here: <https://cloud.digitalocean.com/networking/domains>

The site has been installed as a service so that if the machine is rebooted, it will start up again on its own.

**The GitHub repository can be found here:** [**https://github.com/malaclypse/SurveyApp**](https://github.com/malaclypse/SurveyApp)

#### Steps to deploy a new version (after new changes have been committed in the Github repository):

1. Stop the droplet from the Digital ocean dashboard
2. Resize the droplet to “Basic” that costs $10/mo (we will reverse this later)
3. Turn the droplet back on.
4. Login to the droplet/vm with root.
5. Navigate to /var/www/survey-app
6. Delete the /publish directory
7. “git pull” to get the new code changes
8. If there are any unstaged changes by change in the repo, do “git stash save --keep-index --include-untracked” to remove them
9. Navigate to the /src directory in the survey-app folder
10. Run “dotnet publish SurveyApp.sln -c Release --self-contained false -o ../publish”
11. Restart the service, “systemctl restart survey.service”
12. Verify that the website runs ok (delete browser cache just in case).
13. Stop the droplet and resize it back to the most “Basic” plan that costs $5, then turn it back on.

## 

## Team Tasks

Note: the tasks around the implementation of a single endpoint are:

* Create a new service interface in the SurveyApp.Services if it doesn’t already exist, and define methods that are needed for the action.
* Create a new service class that implements that interface if it doesn’t already exist.
* Implement the methods that are required for the action (this is where the actual database queries happen and where data is verified, and errors are thrown if it’s not valid data).
* If necessary, add methods for translating the data object from the web request to the data objects that are used by the database.
* In the SurveyApp.Web project, create a new Controller class, if it doesn’t already exist.
* In the specific controller define the endpoint methods that are needed (this is where the URL of the endpoint is specified, and what actions are needed). One controller method may need more than one of the services previously defined.
* In the Startup class, the created services are listed for the dependency injection to work successfully.

|  |  |
| --- | --- |
| Team member | Task |
| Deia Georgieva | Overall project setup  Frontend  Admin panel |
| Tsvetelina Ivanova | Backend:   * Endpoint for returning all groups currently configured in the db, in order to visualize them as groups in the UI * Endpoint for updating the survey, for example when the survey is submitted, and it needs to be marked as completed * Endpoint for returning a specific survey after it’s been created by id |
| Samuela Nenova | Backend:   * Endpoint for returning the texts for the specific survey variant * Endpoint for returning all created surveys * Endpoint for creating a new survey, with a new unique variant for the user, so that each variant is provided only once. |
| Konstantin Terziev | Backend   * Endpoint for returning all mappings for a survey * Endpoint for returning a single mapping by id * Endpoint for deleting a mapping by id * Endpoint for creating a new mapping, that will also delete the previous mapping for this text that exists |