Technical Specification Document

This document is intended to help future development of the Dura Vermeer RIE tool. My hope is that this document will provide a starting point for changes or upgrades that need to be made, by specifying how to run and deploy it, how it's structured, what techstack was used in development, and if all else fails how to contact the original author.

Techstack

The following tools and programs were used in developing the tool.

Programming languages / frameworks / libraries

- Typescript 5.5.2
- Angular 18.2.11
- Angular Material 18.2.11
- angular-cli-ghpages 2.0.1

Tools

- Visual Studio Code
- InteliJ IDEA 2022.3.3
- Microsoft Office 365

Others

- GIT
- GitHub
- GitHub Pages

Local setup

To run this tool locally, you first need to have installed GIT, Typescript, and Angular. Then you can follow the following steps:

1 Clone the repository
You first need to have the code stored on your device. This can be done by a simple git clone:

```
>>> git clone https://github.com/duravermeer-windesheim/RIE.git
```

2 Install packages
If the repository is cloned, navigate into the folder, and open a new terminal. There you can enter the following command to install the packages:

```
>>> npm install
```

3 Run it
After installing the packages, you can always run it by simply entering the command

```
>>> ng serve
```

This will serve the tool on localhost:4200, which you can visit in your browser of choice.

Hosting

The project is hosted using GitHub pages. This is a great way to host applications without additional cost. The files that get served when you are online, can be found on the gh-pages branch. That means that gh-pages is not allowed as a branch name.

You can deploy it using the following command:

```
>>> npm run deploy
```

This is a shorthand of the full command, which can be found in package.json:

```
>>> ng deploy –cname=ehbort.nl –repo https://github.com/duravermeer-windesheim/RIE
```

If the domain name changes, you can change the CNAME in this command as well for npm run deploy to work again.

Version management

Version management is done by hosting a GIT repository on GitHub. Branches use kebab-case, and on pull requests, the commits are squashed into 1.

The repository is inside an organization, so it allows very specific permissions. If you need permissions, please contact me and I'll be happy to help.

Code organization

On the right in figure 1, you see the full file structure, inside the src folder.

The src folder contains an app folder, and an environments folder.

Environments folder

The environments folder only contains environments.ts, which has a lot of configuration, including the following configuration items:

- googleSpreadsheet,
- adviceThresholds
- colorConfig
- sheetNames

Each item has a comment explaining what it means.

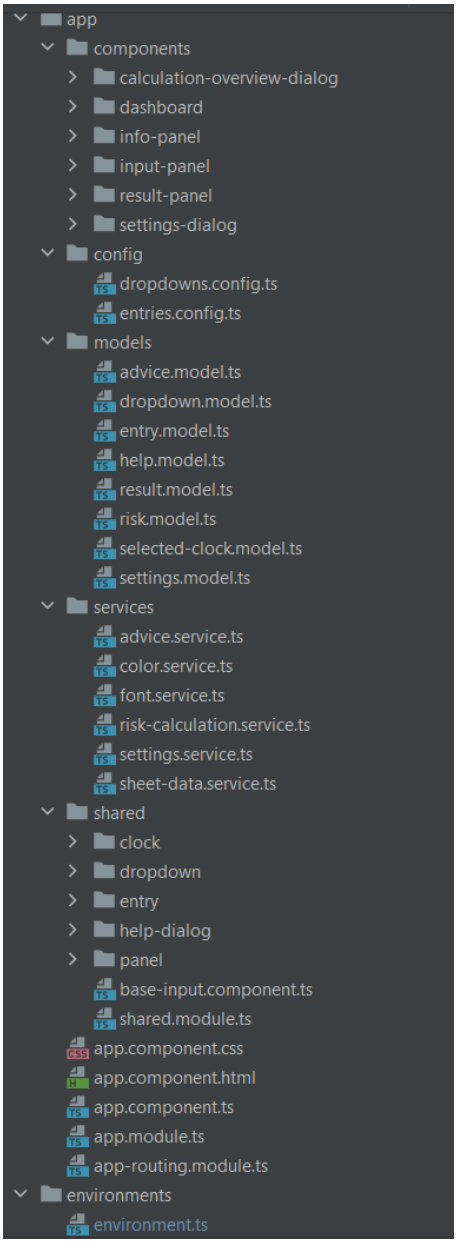

Figure 1: File structure

App folder

In the app folder, the actual application lives.
It is divided into components, config, models, services, and shared components.

Components folder

In components, you will see the code that handles the page itself. Dashboard is a container for the 3 panels (info-panel, input-panel, and result-panel), and for the settings-dialog. calculation-overview-dialog comes from result-panel. I won't go into the details of each component in this document, but the components contain comments to explain how things worked.

Shared folder

The shared folder contains a couple of components that get used multiple times, and conditionally. These include the clocks, help dialogs, and the generic dropdowns and entries. These two input components both inherit base-input.component.ts.

They both accept a config object that sets things like labels, whether they are disabled, their placeholders etc. Those config objects can be found in the config folder. At the time of writing, I am thinking of restructuring that part.

Models folder

The models folder is pretty straight-forward. It contains all models used in the application. An example is risk.model.ts, which houses all models related to risks. (A detailed explanation of what a RiskScoreGroupCollectionModel is, will follow later).

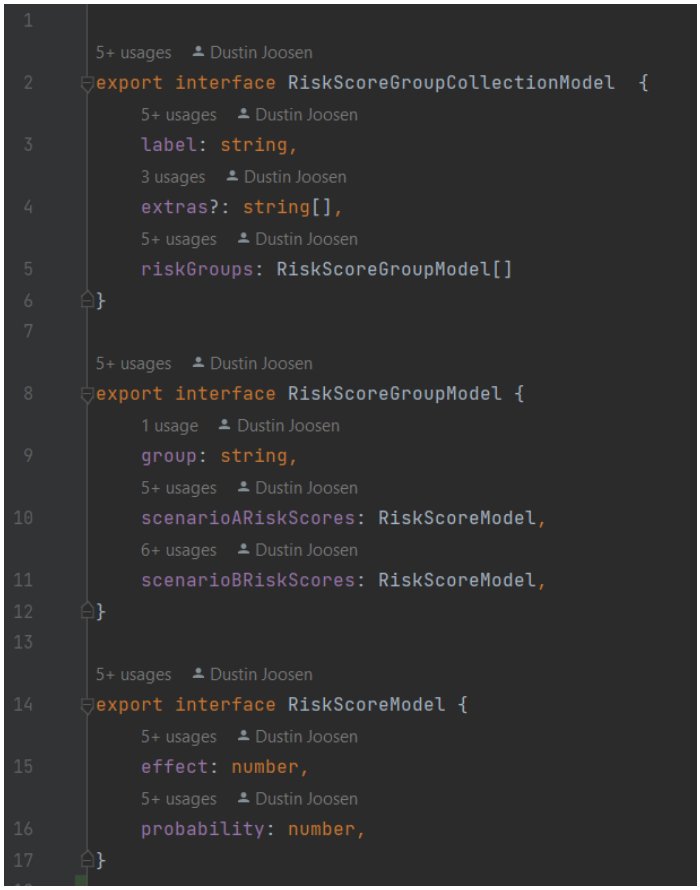

Figure 2: Risk.model.ts

Services folder

The services folder contains all services used. I will quickly walk through each of the services:

advice.service.ts: Handles creating an advice. The only public method is getAdvice, which results an AdviceModel, and takes in a ResultModel. This service will mostly be called directly after the risk calculation service gives you its resultmodel. These models can of course be found in the Models folder.

color.service.ts / **font.service.ts**: Handles custom colors, and font sizes. The user can change settings for the primary and second color, and the main font size. These are handled by these two services. An initializeColor and initializeFont method exist, and they are both called in app.component.ts, on the site loading for the first time. These methods then set the colors accordingly for the entire site.

risk-calculation.service.ts: Handles the actual calculation of the risk scores. It gets a CalculationModel (all selected data by the user In the input panel, risk type, measures, frequencies etc.). It returns the calculated risk scores for each risk group in both scenario's.

settings.service.ts: Related to the color and font services. It stores and loads these values from the localStorage. The key used, can be found in the environment.ts file.

sheet-data.service.ts: Retrieves information from the google sheets API. It's internal method *private async getSheetData(sheet: string): **Promise**<[...string[]][]> {* handles this retrieval, while the other methods in the file use it and handle parsing it.

Risk models

A messy structure is used for risks, As seen in figure 2. This chapter handles what each of the 3 models mean. The structure houses the following data:

Aanrijding VKM-ploeg niet vluchtstrook		Maatregelen: Flitscamera's pl... Drempels neerle... Snelheid verlagen		
Automobilist				
Omwonende				
VKM ploeg	15	6	15	6
Wegwerker				

Figure 3: Risk

Now, from the bottom to the top:

A RiskScoreModel is a combination of 2 of the numbers. For example effect: 15, and probability: 6.

A RiskScoreGroupModel is 1 row. It has the group name, scenario A, and scenario B. So group: "VKM ploeg", scenarioARiskScores: the left 15&6, scenarioBRiskScores, the right 15 & 6.

A RiskScoreGroupCollectionModel is the entire risk. It has the label at the top "Aanrijding VKM-ploeg niet vluchtstrook", the extra (Measures), and an array of the rows. This entire thing is considered 1 risk.

The sheet-data.service gets and parses the data like this. It returns an array of risks, (RiskScoreGroupCollectionModel[]). The measures use the same structure.

Dependencies

The following angular packages are used by the program:

- "angular-cli-ghpages": "^2.0.1",
- "@angular/material": "^18.2.11",

These two packages are used for deployment and styling respectively. They can be found in the file package.json.

One other dependency is the Google Sheet API. It is used for getting risks and measures, and is required for the tool to work properly.

If it breaks, you can find code pertaining to this in src/app/services/sheet-data.service.ts. There, the method getSheetData handles the retrieval of the sheet data. Here you can update it when the Google Sheet API updates and breaks something.

Contact

Email: dustinjoosen2003@gmail.com