Technische specificaties

Dit document is bedoeld als hulpmiddel bij de toekomstige ontwikkeling van de Dura Vermeer RIE tool. Ik hoop dat dit document een startpunt zal bieden voor wijzigingen of upgrades die moeten worden gemaakt, door te specificeren hoe het oorspronkelijk werd uitgevoerd en geïmplementeerd, hoe het is gestructureerd, welk techstack is gebruikt bij de ontwikkeling en als al het andere faalt, hoe contact op te nemen met de oorspronkelijke auteur.

Techstack

De volgende tools and programma's zijn gebruikt bij de ontwikkeling van de tool.

Programmeer talen / 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

Andere

- GIT
- GitHub
- GitHub Pages

Local setup

Om de tool lokaal te kunnen runnen, moet je zowel GIT, Typescript, and Angular geïnstalleerd hebben. Daarna kan je de volgende stappen volgen:

1 Clone de repository
Als eerst heb je de code lokaal nodig op je apparaat. Dit kan worden gedaan met een simpele git clone.

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

2 Installeer packages
Als de repository gecloned is, navigeer naar die folder, en open een nieuwe terminal. Voer hierin het volgende command in om de packages te installeren.

```
>>> npm install
```

3 Run het
Na het installeren van de packages, kan je het simpel runnen met het volgende command:

```
>>> ng serve
```

Dit served de tool op localhost:4200, waar je in de browser op kan zoeken

Hosting

Het project wordt gehost met GitHub pages. Dit is een makkelijke manier om applicaties te hosten zonder extra kosten. De bestanden die gebruikt worden als je online bent, kunnen gevonden worden op de gh-pages branch. Dat betekent dat gh-pages niet is toegestaan als branchnaam.

Je kunt het online gooien met het volgende command:

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

Dit is een kortere versie, die eigenlijk het volgende command uitvoert, dat gevonden kan worden in package.json:

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

Als de domein naam veranderd, kan je de CNAME in dit command ook veranderen, zodat npm run deploy weer zal werken.

Version management

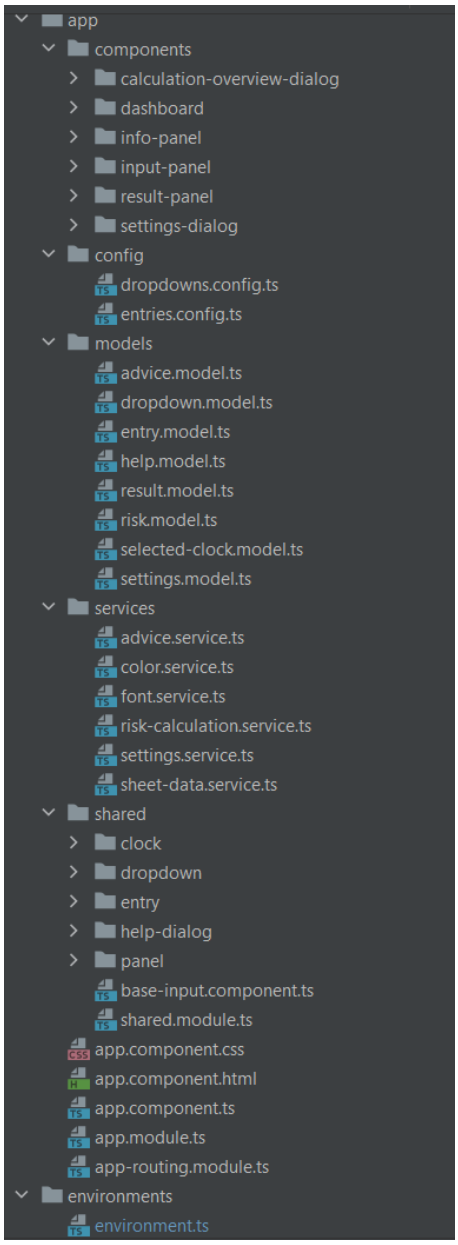
Versiebeheer wordt gedaan door het hosten van een GIT repository op GitHub. Branches gebruiken kebab-case en bij pull requests worden de commits in 1 gesquashed.

De repository bevindt zich binnen een organisatie, dus het staat zeer specifieke permissies toe. Als je deze nodig hebt, neem dan contact met mij op en ik help je graag.

Code organisatie

Rechts in figuur 1 zie je de volledige bestandsstructuur in de src-map.

De src-map bevat een app-map en een environments-map.



Environments map

De map environments bevat alleen environments.ts, dat veel configuratie bevat, waaronder de volgende configuratie-items:

- adviceThresholds
- colorConfig
- sheetNames

Elk item heeft commentaar met extra uitleg in het bestand zelf

App map

In de app-map staat de eigenlijke applicatie.

Deze is onderverdeeld in componenten, config, models, services en gedeelde componenten.

Components map

In componenten zie je de code die de pagina zelf afhandelt. Dashboard is een container voor de 3 panelen (info-paneel, input-paneel en resultaten-paneel), en voor de instellingen-dialoog. calculation-overview-dialog komt uit resultaten-paneel. Ik zal in dit document niet ingaan op de details van elke component, maar de componenten bevatten commentaar om uit te leggen hoe dingen werkten.

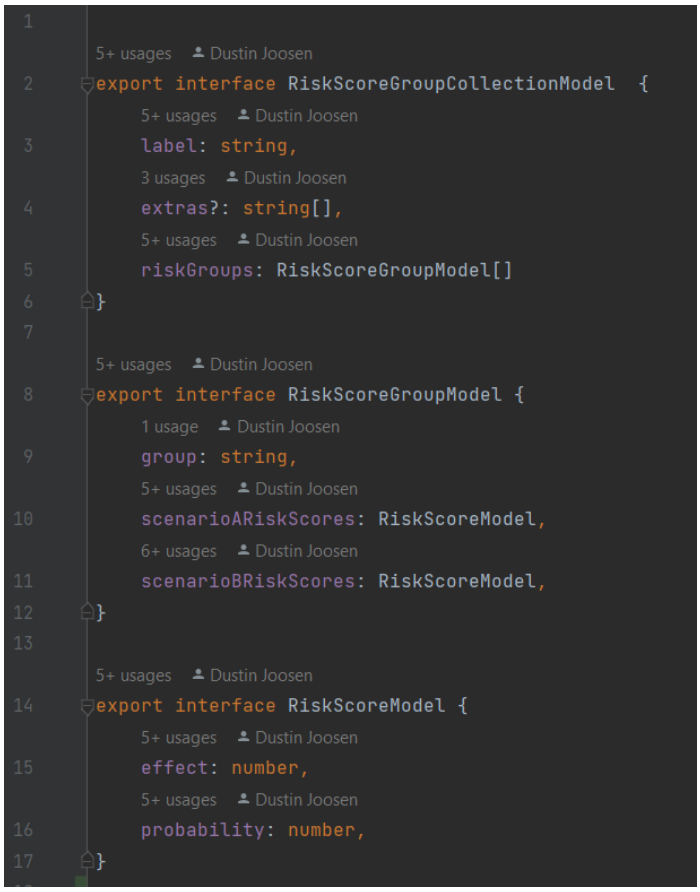
Shared map

De gedeelde map bevat een aantal onderdelen die meerdere keren en voorwaardelijk worden gebruikt. Dit zijn onder andere de klokken, helpdialogen en de algemene dropdowns en invoer. Deze twee invoercomponenten erven allebei van base-input.component.ts.

Ze accepteren allebei een config-object dat dingen instelt zoals labels, of ze uitgeschakeld zijn, hun plaatshouders enz. Deze config objecten kunnen gevonden worden in de config map. Op het moment van schrijven denk ik erover om dat deel te herstructureren.

Models map

De models map is vrij eenvoudig. Deze bevat alle modellen die in de applicatie worden gebruikt. Een voorbeeld is risk.model.ts, waarin alle modellen met betrekking tot risico's staan. (Een gedetailleerde uitleg van wat een RiskScoreGroupCollectionModel is, volgt later).



Figuur 1: Risk.model.ts

Services map

De map Services bevat alle gebruikte services. Ik zal alle services doorlopen:

advice.service.ts: Handelt het maken van een advies af. De enige openbare methode is getAdvice, die een AdviceModel oplevert en een ResultModel inneemt. Deze service wordt meestal direct aangeroepen nadat de risk-calculation service het resultaatmodel heeft gegeven. Deze modellen kunnen natuurlijk worden gevonden in de map Models.

color.service.ts / **font.service.ts**: Verwerkt aangepaste kleuren en lettergroottes. De gebruiker kan instellingen wijzigen voor de primaire en secondaire kleur en de lettergrootte. Deze worden afgehandeld door deze twee services. Er bestaan een

methode initializeColor en initializeFont, en ze worden allebei aangeroepen in app.component.ts, wanneer de site voor het eerst wordt geladen. Deze methoden stellen dan de kleuren in voor de hele site.

risk-calculation.service.ts: Handelt de daadwerkelijke berekening van de risicoscores af. Het krijgt een CalculationModel (alle door de gebruiker geselecteerde gegevens in het invoerpaneel, risicotype, maatregelen, frequenties enz.) Het geeft de berekende risicoscores voor elke risicogroep in beide scenario's terug.

settings.service.ts: Gerelateerd aan de kleur- en lettertype services. Deze service regelt het opslaan en laden vanuit de localStorage. De gebruikte key kan worden gevonden in het bestand environment.ts.

sheet-data.service.ts: Haalt informatie op uit de google sheets API. De interne methode *private async getSheetData(sheet: string): Promise<[...string[]][]> {* handelt dit ophalen af, terwijl de andere methoden in het bestand het gebruiken en het parseren afhandelen.

Risk modellen

Voor risico's wordt een rommelige structuur gebruikt, zoals te zien is in figuur 2. Dit hoofdstuk behandelt wat elk van de 3 modellen betekent. De structuur bevat de volgende gegevens:

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

Figuur 2: Risico

Vanaf beneden naar boven:

Een RiskScoreModel is een combinatie van 2 van de getallen. Bijvoorbeeld effect: 15, en waarschijnlijkheid: 6.

Een RiskScoreGroupModel is 1 rij. Het heeft de groepsnaam, scenario A en scenario B. Dus groep: “VKM ploeg”, scenarioARiskScores: links 15&6, scenarioBRiskScores: rechts 15 & 6.

Een RiskScoreGroupCollectionModel is het hele risico. Het heeft bovenaan het label “Aanrijding VKM-ploeg niet vluchtstrook”, de extra (Maatregelen) en een array van de rijen. Dit geheel wordt beschouwd als 1 risico.

De sheet-data.service haalt de gegevens op en verwerkt ze als volgt. Hij retourneert een array van risico's (RiskScoreGroupCollectionModel[]). De maatregelen gebruiken dezelfde structuur.

Dependencies

De volgende angular-packages worden door het programma gebruikt:

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

Deze twee pakketten worden gebruikt voor respectievelijk deployment en styling. Ze kunnen worden gevonden in het bestand package.json.

Een andere afhankelijkheid is de Google Sheet API. Deze wordt gebruikt voor het verkrijgen van risico's en maatregelen en is nodig om de tool goed te laten werken.

Als het breekt, kun je de code hiervoor vinden in src/app/services/sheet-data.service.ts. Daar zorgt de methode getSheetData voor het ophalen van de sheetgegevens. Hier kun je het bijwerken als de Google Sheet API updates uitvoert en iets verbreekt.

Contact

Email: dustinjoosen2003@gmail.com