Animated Sky

Project Report

Twan Boeve (s2377675)
Pim Mulder (2240084)
Patrick van Oerle (s2010267)
Wout Velthuis (s2335255)
Jelle van den Wijngaard (s1574922)

Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente
202001048 – Design Project Module
Supervisor: dr. D. Bucur
April 24, 2022

Contents

1 Introduction **3**
1.1 Project Goal . . . 3
1.2 Intention of the Report . . . 3

2 Project Requirements **4**
2.1 Functional Requirements . . . 4
2.2 Quality Requirements . . . 4
2.3 Constraints . . . 4
2.4 Achieved Project Requirements . . . 4

3 System Overview **6**
3.1 Goals of Design Structure . . . 6
3.2 Project Structure . . . 6
3.2.1 Index . . . 7
3.2.2 App . . . 7
3.2.3 Settings . . . 9
3.3 Data Preparation . . . 10
3.3.1 Nodes . . . 10
3.3.2 Edges . . . 12
3.4 Miscellaneous Components . . . 14
3.4.1 ColourProvider . . . 14
3.4.2 DataProvider . . . 14
3.4.3 Internationalization . . . 14
3.4.4 Icons . . . 14
3.4.5 LocalStorage use . . . 14
3.5 Implementation . . . 15
3.5.1 Forms . . . 15
3.5.2 Icons . . . 15
3.5.3 Modals . . . 16
3.6 sidemenu . . . 16
3.7 Stars . . . 17
3.8 Pages . . . 19
3.9 Utils . . . 19

4 Testing Overview **21**
4.1 Unit Tests . . . 21
4.2 Hands-on Testing . . . 21

5 Visual Design **22**
5.1 The Main Screen . . . 22
5.2 Zoomed-out view . . . 23
5.3 Cultures . . . 23
5.4 Presets . . . 25
5.5 Settings . . . 25
5.6 Colourblind palettes . . . 27

6 Limitations And Bugs **28**
6.1 Culture Selection . . . 28
6.2 Zoomed-in Star Display . . . 28
6.3 Zoomed-out Star Display . . . 29
6.4 Incorrect drawing of the edges . . . 29

7 Future Work **30**
7.1 Caching . . . 30
7.2 React functions . . . 30
7.3 Languages . . . 30
7.4 Information about the cultures . . . 30
7.5 Option for joint edges . . . 30
7.6 Tutorial . . . 30
7.7 Edge highlighting . . . 30
7.8 Import button . . . 31

8 Risk Analysis **32**
8.1 System Design . . . 32
8.2 Data Processing . . . 32
8.3 Development Plan . . . 32
8.4 Deployment Plan . . . 32

9 Manual **34**
9.1 Side Menu . . . 34
9.2 Star Display . . . 34
9.3 Settings Menu . . . 35
9.4 Setup . . . 35

10 Project organization **36**
10.1 Group Process . . . 36
10.1.1 Group dynamic . . . 36
10.2 Planning . . . 36
10.3 Individual Process . . . 36
10.3.1 Twan Boeve . . . 36
10.3.2 Pim Mulder . . . 36
10.3.3 Patrick van Oerle . . . 37
10.3.4 Wout Velthuis . . . 37
10.3.5 Jelle van den Wijngaard . . . 37
10.4 Contact . . . 37
10.4.1 Client . . . 37
10.4.2 Supervisor . . . 38
10.4.3 Combination of Client and Supervisor . . . 38

11 Presentation **39**

12 Poster **44**

1 Introduction

1.1 Project Goal

Throughout history, cultures from all over the world have come up with their own way of identifying the stars decorating the night sky. This unique identification lead to each culture having their own variant on constellations. As with any other cultural feature, this heritage is often overlapping between cultures. However, it may be that two cultures very similar in the used constellations tell complete opposite stories of how the constellations came to be. Moreover, constellation cultures vary in size for several reasons, including data loss over time by loss of cultural artifacts, or by dominant cultures erasing that heritage. Next to that, cultures could also increase their amount of identified constellations by seeing more of the world by travelling and thus increasing their need for this navigational tool.

As a consequence of the interesting difference, this project strives to build an interactive application to visualize the different sky cultures in the Stellarium[1] sky cultures plugin. The aim is to provide a visually pleasing and efficient way of comparing the constellations around certain stars envisioned in different cultures.

The difference between this project and existing applications for exploring the sky is that it focuses on comparing the constellations themselves rather than representing the entire sky as accurately as possible. In other words, visualizing the difference between constellations from different cultures is the primary focus of this project. This is different from other projects, such as Stellarium, where the focus is on having a perfectly accurate representation of the current state of the sky. Consequently, competitors support no feasible way of comparing many sky cultures simultaneously yet. It is believed that this is partly due their user interface not being designed for that purpose.

The project is aimed at the general public, as the application is intended to be used by anybody interested. It should be easy to navigate and understand on any PC web browser.

1.2 Intention of the Report

This report consists of two parts, the product explanation and the project organization. The first part will compare the desired product with the achieved product and with which technicalities that was fulfilled. Moreover, it will explain how the quality was guaranteed in the form of the testing plan. Finally, it will go into depth about the user interface choices and how that could bring implied limits to the product. Added to those implied limits will be explained what other limits the product has paired with the risks the product takes.

Then, it will be explained in what way the project organization benefited or hurt the final result. It will be reflected how the group dynamic and planning was, and how each individual of the group has contributed and what they have learned from the process. Finally, the contact with both supervisor and client will be reflected upon.

[1] https://github.com/Stellarium/stellarium-skycultures

2 Project Requirements

2.1 Functional Requirements

Functional Requirement No.	Description
FR 1	As a user I want to have a general overview of which popular stars are used in which constellations
FR 2	As a user I want to have a view with all stars
FR 3	As a user I want to be able to see the constellations of a culture
FR 4	As a user I want to be able to see all stars included within constellations of the culture
FR 5	As a user I want to be able to select multiple cultures
FR 6	As a user I want to be able to compare the constellations of a culture
FR 7	As a user I want to be able to select edges to see all constellations including that edge
FR 8	As a user I want to be able to select stars to see all constellations including that star
FR 9	As a user I want to be able to get information about a star on hover or click of the star
FR 10	As a user I want to be able to get information about a constellation on hover or click of the constellation
FR 11	As a user I want to be able to change the colours to colourblind friendly colours
FR 12	As a user I want to be able to have a zoomed-out version of the map
FR 13	As a user I want to be able to have a zoomed-in version of the map
FR 14	As a user I want to be able to have a preset of a set of selected and unselected cultures
FR 15	As a user I want to be able to create my own presets of selected and unselected cultures
FR 16	As a user I want to be able to store my presets for later use
FR 17	As a user I want to be able to export/import settings and presets
FR 18	As a user I want to have the program in my own language

Table 1: Functional Requirements

2.2 Quality Requirements

Quality Requirement No.	Description
QR 1	The program should load within 5 seconds on modern systems
QR 2	The program should be easily navigable by first time users
QR 3	The visualisation should be clear even when selected many cultures
QR 4	The program should be usable by all ages seven and older

Table 2: Quality Requirements

2.3 Constraints

Constraint No.	Description
C 1	The program only works on desktop and laptop web browsers
C 2	The program only works on up-to-date major web browsers

Table 3: Constraints

2.4 Achieved Project Requirements

Tables 1, 2, and 3 display the requirements that were set up at the start of the project. In the delivered product, almost each has been met. That said, two functional requirements could not be included: Throughout the development process, it became apparent that the way constellations

are visualized has next to no support to highlighting an edge on the screen - the precise reasoning is explained in Chapter 3. Therefore, unfortunately, functional requirement 7 had to be disregarded.

Functional requirement 10 also proved to be unnecessary. During the process, it became clear that this requirement was, in fact, not wanted. Namely, having a popup on both nodes and edges of constellations would prompt the user too much information, thus making the application close to impossible to use. Ergo, after consideration within the project group, it was settled not to include this feature.

Besides, due to time constraints one of the quality requirements could not be met: It was intended that the user would be introduced to the program via a tutorial on the page explaining the functionality of the webapp. However, the aforementioned time constraints combined with the fact that this was not a priority of the client saw for this feature to be dropped. Moreover, this pill was sweetened by the inherent intuitive design of the website.

The constraint requirements were also used with developing the project, which helped with keeping the scope in check.

3 System Overview

3.1 Goals of Design Structure

The intention of this project is to hand over the code to another party, after which they will take care of further maintainability and deployment. Ergo, the source code of the project needs to be as maintainable, comprehensible, and efficient as possible. This is achieved if another developer could understand the logic behind the project by reading this chapter. Therefore, this chapter will explain the system design combined with the reasoning behind that and the way the components and functions have been implemented.

3.2 Project Structure

This project is build using React[2], this implies that the project is build up of recyclable components. In this subsection, it will be explained how each component relates to each other and how information is passed throughout the system.

```
src
├── components
│   ├── forms
│   │   ├── generalDropdown
│   │   ├── generalSwitch
│   │   └── generalSlider
│   ├── icons
│   │   └── Icon.jsx
│   ├── modals
│   │   ├── AddModal.jsx
│   │   ├── GeneralModal.jsx
│   │   └── PresetsModal.jsx
│   ├── sidemenu
│   │   ├── cultures
│   │   │   ├── Culture.jsx
│   │   │   └── Cultures.jsx
│   │   ├── presets
│   │   │   ├── Preset.jsx
│   │   │   └── Presets.jsx
│   │   ├── Header.jsx
│   │   └── SideMenu.jsx
│   └── stars
│       ├── LoadingStars.jsx
│       ├── Stars.jsx
│       ├── ToolTip.jsx
│       └── ZoomedOutNode.jsx
├── pages
│   ├── App.jsx
│   └── Settings.jsx
├── utils
│   ├── colourProvider
│   │   ├── colourProvider.js
│   │   └── palettes.js
│   ├── dataProvider
│   │   ├── data
│   │   └── DataProvider.js
│   └── errors
├── general.css
├── i18n.js
└── index.jsx
```

Figure 1: File Structure of source code

[2]https://reactjs.org/

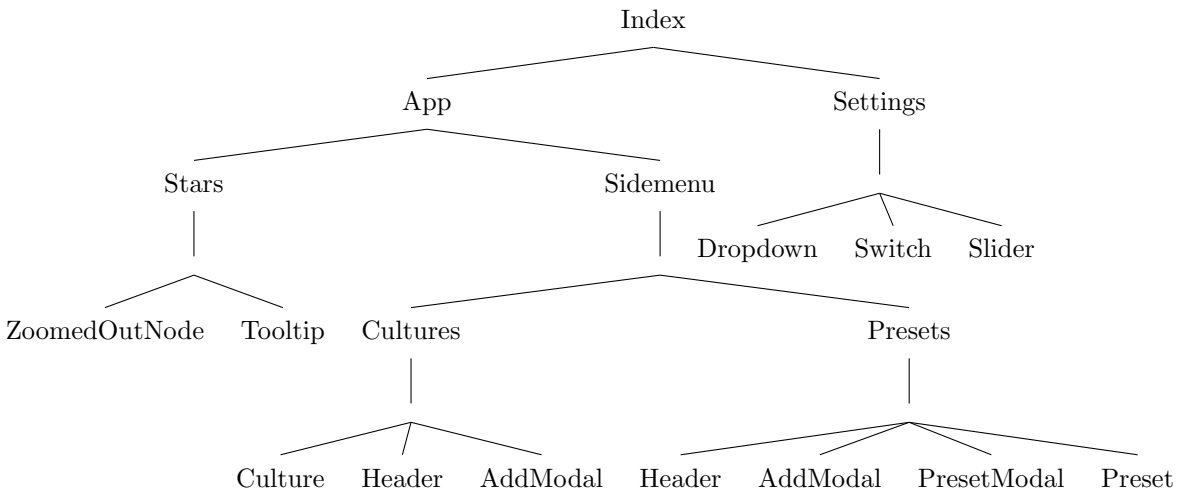


Figure 2: Overall System Structure

Figure 1 displays the file structure of the source code. To create a more concise overview of which component uses what other components, Figure 2 has been added.

3.2.1 Index

As for most projects, the rendering starts with an index file. For this project, the only functionality for it is to render the react project on the screen, and to route the user to the correct web-page. Routing is done using the react-router-dom[3] package. By using this, the user is told via the taskbar on what page they are, and they could skip specific screens by going to the route immediately. From thereon, the router calls two pages contained in the pages directory - as shown in Figure 1.

3.2.2 App

Since the App is the prime function of the system, that will first be explained: App is the landing-page of this web-app, it is the page that is able to compare constellations and allows the user to modify their selected cultures. To better visualize the following explanation, the reader could first take a look at the mock-ups listed in Chapter 5.

The web-app is split up into two screens, a collapsible side menu and a canvas on which the constellations are rendered. Essentially, constellations can be seen as graphs, where the stars form the edges and the edges combine to form the constellation. Therefore, a graph visualization library named react-graph-vis[4] is used to display the comparison.

3.2.2.1 Stars

This graph visualization library makes use of an HTML canvas to visualize the provided nodes and edges. The nodes therefore each have an x- and y-coordinate to position them on the canvas. The edges are identified by providing a from-id and a to-id. Consequently, for the nodes to be identifiable, they are required to also have a unique id.

These nodes and edges are provided to the component by a helper class named DataProvider, which is explained in subsection 3.4. As per the functional requirements listed in Table 1, the user should have two possible star visualizations: zoomed-in and zoomed-out. Since the former displays all stars to the user, it per definition requires the canvas to render many more nodes. Therefore, two data providers are used within the component; one for the zoomed in view and the other for the zoomed out view. The difference being that, the zoomed in view provides data in a less wide radius, thus reducing the rendering load. However, that does not imply their may be a

[3] https://www.npmjs.com/package/react-router-dom
[4] https://www.npmjs.com/package/react-graph-vis

waiting time in getting the data. Therefore a loading screen has been added which makes use of the package react-content-loader[5].

Moreover, selecting this data provider is done by storing the zoom position of the user. If a predetermined threshold is crossed, the data provider is swapped and the view is updated. Additionally, the component ensures the styling of the nodes and edges is matched with those in Chapter 5. The implementation of the styling is explained in Section 3.5. If the zoom position exceeds the range of how far a user can zoom in or out, their position is corrected for them to prevent them from being made aware of the data provider's radius.

Next to the zoom position of the user, the coordinates of the centre of the view position are being stored. Whenever the user updates those data, the internal state is updated and the data are re-fetched relevant to that x- and y-coordinate, i.e. being within the aforementioned radius.

Besides dealing with the data, the component takes care of other visual features: Small ones such as a smooth transition when collapsing the side menu, or zooming in to a star on click in the zoomed out screen, but also bigger ones like the tool tips. The latter is regarded as too large to be included within the Stars component itself. Consequently, it is moved to a separate component, benefiting maintainability. The shear functionality of that component is to provide an HTML element to each edge, however for that JSX needs to be converted to an HTML element increasing the complexity - as for any other component, the details are explained in Section 3.4.

Finally, the Stars component relies on the provided culture, these are given to it via App. The App component also contains the methods to operate on this culture set. These are then passed down to the side menu which can then provide them to Cultures and Presets to operate on them. Whenever the cultures are updated, only the data is fetched again to minimize re-rendering of components.

3.2.2.2 Sidemenu

The side menu distinguishes two selection screen: Modifying cultures and modifying presets. In this context, modifying cultures is either selecting, deselecting, listing or unlisting. Moreover, modifying presets means, selecting, deselecting, adding or removing them. Table 4 displays what the expected behaviour is for each mutation in what category. This is guaranteed by the passed down functions to the Sidemenu component.

Category	Mutation	Expected Behaviour
Culture	Selecting	The constellations of this culture are now displayed and the styling in the side menu is updated to be more outstanding
Culture	Deselecting	The constellations of this culture are not displayed anymore and the styling in the side menu is updated to be less noticeable
Culture	Listing	The culture is added to the side menu for the user to select or deselect
Culture	Unlisting	The culture is removed from the side menu completely
Preset	Selecting	The cultures of the preset are put in as selected, or listed cultures, and the preset is highlighted to show it is selected
Preset	Deselecting	The cultures of the preset remain selected, but the preset is not highlighted any more
Preset	Adding	A preset is added to the side menu with a given name, on select it will select and list the given cultures
Preset	Removing	The preset is removed from the side menu and can no longer be retrieved any more.

Table 4: Expected behaviour on culture or preset modification

Since the user will often make calls such as unlisting, removing or deselecting in quick success, these calls will be debounced to the stars component. Meaning, whenever a user repeats a call within 500 ms the timer will restart and only after this time has elapsed will the component start re-fetching its data. Note that, this is not the case for the side menu itself. There, the internal cultures are immediately altered to give the user a sense of responsiveness.

[5]https://www.npmjs.com/package/react-content-loader

Next to the functional part, the side menu has visual functions to provide a useful interface to the user to operate on the selected and listed cultures. Since the side menu is provided with which cultures are selected and listed, and which presets exist and are selected - maximum of one, it can easily display them. This is done by invoking a separate component, Culture or Preset depending on which task to do. This decision is made as the component consists of two parts, the indicator of the culture, and a label on hover. To ensure this label is also displayed when the user hovers on text, this variable is upkept within the sub-component. Besides, it creates smaller components increasing comprehensibility. In the case of the sub-component Culture, a colour indicator is put to the left of the div, the colour to this square is provided by the ColourProvider explained in Section 3.4

That said, this list is only able to let the user select and deselect cultures, or unlist or remove cultures or presets. Adding a preset or listing cultures is done via a popup screen accessible via the header. The header is provided to both classes with a title and what functions it can serve. In the case of the culture only a setting, modifying the listed cultures, but in the case of the presets three functions are provided: Creating a new preset with nothing pre-selected, with the current selection pre-selected. Based on the provided functions the Header component settles which icons to display.

3.2.2.3 Modals

Following the provided functions the Header component can summon two types of modal, either an AddModal or a PresetsModal. Either are build upon a GeneralModal, this component takes care of the styling of the modal as well as the closing and opening animation. The AddModal can add cultures or presets to the list in the side menu. In either case, the list of selected and listed cultures is maintained and on confirm a provided function is performed. When selecting cultures this insists on replacing the current array of selected and listed cultures. In the case of presets, this implies, appending the array of presets with the newly selected preset. Note, whenever a culture is selected it will always be listed as well.

To make looking through the cultures easier, a search bar is put within the AddModal. The input of that acts as a filter, currently, it only looks whether a culture name starts with the provided input, later it could be modified to increase intuitiveness. In the situation of adding a preset, an extra input bar will appear. The function of that input bar is to allow the user to provide a name to the preset, if that name is already existing or empty, the addition is cancelled.

With all the modifications listed in Table 4 it may be expected no extra modal is needed. However, that would leave the user with no possibility to get an overview of the contents of each preset. Therefore, the PresetsModal has been added. The task of the component is to list each preset paired with what cultures will be listed. Additionally, they will be highlighted if they are selected by default. On hover of the preset, the user can decide to remove it by clicking on the cross that appear with that interaction. This functionality takes inspiration from the Culture or Preset component but is kept within the same file as it has less functionality, for example it is not required to display a colour tag or change styling in any scenario.

3.2.3 Settings

Several options within the system are modifiable, all these options are listed in settings and can be changed there as well. The settings categorize in four groups: Selecting from a group of options, selecting a number, toggling a feature or having buttons to actions. The latter happens only once in the settings page, and thus it has been chosen to limit that to standard JSX code. However, the other three categories are each generalized to standardize the styling and expected behaviour.

This results in three general components, as shown in Figure 2. Each component is given a default value, a label, and what should happen on change of the data. In the case of the Dropdown, the options are given to it, and in the case of the Slider the minimal value and maximum value are provided. Which then limits the slider and the input field. To take some of the effort away of the styling of all components, the library antd[6] and react-bootstrap[7] were used. Specifically, the slider and switch make use of antd's slider and switch, and the dropdown is build up from bootstrap's dropdown. Additionally, the ColourProvider is given to these components to keep them in line

[6] https://www.npmjs.com/package/antd
[7] https://www.npmjs.com/package/react-bootstrap

with the selected colour palette.

For each component, changing the value comes down to changing the local storage of the webpage. This ensures that the data of the user is stored, even when leaving the page. Therefore, the user will find an increase in continuity of the program. Added to that is the fact that some settings relate directly to the settings page itself: Language and colour palette. Consequently, that information is stored in the state to update the page to adhere to the user preference on change of those settings.

Finally, if the user desires to export their preference, that is achievable via the export settings button. This is done by pressing the button after which the local storage is exported to a JSON-file. Meaning: The current settings are downloaded, the currently listed and selected cultures, and the created presets of the user. When importing the file, the state of the settings page is updated and the local storage is overridden. Ensuring that the file is actually a settings file is done by checking whether the provided JSON limits its keys to those needed by the system.

3.3 Data Preparation

Gathering the data used by the DataProvider is done with a pair of Python scripts. The reason Python is used for this is because it is an easy-to-use programming language when it comes to reading and writing files. Both scripts can be found in the 'data_gathering' folder in the main directory of the project.

3.3.1 Nodes

The first of these scripts prepares the so-called 'nodes', which are essentially the stars that can be found in the Star Display of the system. At first, the script was relatively simple since all it did was read data from the HYG Database[8], filter this data and write it to a new file. This seemed like it was all that was needed since it contained the HIP number (used as an index number), the proper name (if the star has one), the Right Ascension and Declination (converted to x- and y-coordinates by the script), and finally the Magnitude of almost all stars used by the system; for some stars that were not in the database, the data had to be added manually.

However, there was one small problem with this data. Since it did not contain proper names for most of the stars, it was not possible to distinguish them in the system, making it difficult for the user to compare and recognize certain constellations. This meant that also for these stars data had to be gathered. For this, the SIMBAD Astronomical Database[9] was used, which has the capability of searching stars by their HIP number with the use of a URL. This database contained names for every single star, but there is one problem with it: in order to be able to retrieve the star name, the entire web page has to be loaded by the script for every single star, and then the star name has to be filtered out. This makes the script take very long to finish, but it seemed to be the only possible strategy to retrieve star names for every single star we used.

The final structure of the nodes JSON file can be found in Figure 3 below.

```
{
    {node HIP number}: {
        "label": {node label},
        "x": {calculated x-coordinate},
        "y": {calculated y-coordinate},
        "magnitude": {node magnitude}
    },
    ...
}
```

Figure 3: Structure of the nodes JSON file

[8] https://github.com/astronexus/HYG-Database
[9] http://simbad.cds.unistra.fr/simbad/

In Figure 4 below an example from the final result can be found.


```
{
  "55203": {
    "label": "ksi UMa",
    "x": 5291,
    "y": 3249,
    "magnitude": 100
  },
  "10234": {
    "label": "63 Cet",
    "x": 9087,
    "y": 5102,
    "magnitude": 95
  },
  ...
}
```


Figure 4: An example from the final nodes JSON file

3.3.2 Edges

The second script prepares the 'edges', which are the connections between stars in a constellation. The data used for this is from the Stellarium skycultures database[10] provided by the client of the project. This database contains culture names, named constellations in each culture, and edges in each constellation using the HIP number of the stars.

The Python script goes through the data of each culture one by one, converting the data into a JSON file with the structure found in Figure 5 below.

```
{
    {culture name}: [
        {
            "id": {constellation id},
            "label": {constellation label},
            "edges": [
                {
                    "from": {HIP number of first star},
                    "to": {HIP number of second star},
                    "weight": {weight of edge if given}
                },
                ...
            ]
        },
        ...
    ],
    ...
}
```

Figure 5: Structure of the edges JSON file

[10] https://github.com/Stellarium/stellarium-skycultures

In Figure 6 below an example from the final result can be found.


```
{
  "anutan": [
    {
      "id": "CON anutan 001",
      "label": "The Bamboo",
      "edges": [
        {
          "from": "112623",
          "to": "112122",
          "weight": ""
        },
        {
          "from": "112122",
          "to": "110997",
          "weight": "thin"
        },
        ...
      ]
    },
    ...
  ],
  ...
}
```


Figure 6: An example from the final edges JSON file

3.4 Miscellaneous Components

Next to the rendered components, a few other components have been mentioned that either keep track of information for consistency, or that provide information which is too complex to handle within those components. In this section these features will be explained.

3.4.1 ColourProvider

As per FR 11 in Table1, the user should be able to select from different colour palettes. This is achieved by the use of a colour provider, which provides general styling colours such as background colours, font colours, and border colours, but also the colours of the edges between the stars.

When creating a new colour provider, it will base its palette on the indicated preference in the settings. In that case, the colour provider internally will keep track of which culture colours have already been used to prevent cultures from sharing duplicate colours. Therefore, the system is limited to having only one colour provider, as otherwise the consistency of colours would be harmed. This instance is created within App and passed down to Stars and Sidemenu which pass it through to the rest of the tree. Using this principle, the Stars component can determine which colour to give to the edges and match that with the indicated colour in the side menu.

3.4.2 DataProvider

One of the biggest parts of this project is the data provider, the prime function is to provide the stars and edges that are within radius of the user's view. The implementation of this is explained in Section 3.5, but in short it comes down to filtering all the stars out that are not within a circle around the center point. Then, whenever an edge is unable to find both stars that edge will not be provided to the user. Though sounding simple, selecting the nodes happens with a linear complexity while also accounting for the map to repeat itself after reaching the end.

Next to providing the nodes and edges, the data provider contains many functions that provide information about the scale of the data, or the view of the user: The ultimate x- and y-values, whether the user is zoomed in or out, etc.. But it also has calls that can tell info about the stars or constellations, e.g. the proper name of a culture instead of the id, the constellations that a star is included in, and providing the name of the star.

Finally, since no continuity between components is needed, this class is called on several places. This is useful for the fact that any odd component can require the data provider to translate star ids or culture ids to a proper translation.

3.4.3 Internationalization

Internationalization is done using i18next[11]. Within the root of the source code, a file is stored with the instructions on startup and the translations of each sentence used in the program. Currently, it only supports Dutch and English as those are languages that can be spoken by the project team. Moreover, on startup the only thing the library does on startup is setting the language to be that of the local storage, or to be English.

3.4.4 Icons

Since the system makes use of several icons, a general component has been written for it. This ensures that the developer has to take little care of the styling as that is included within the component. The only thing that this component requires, is the name of the icon to pick the correct image, the action on click, and the colour provider to assume the correct style. Additionally, a classname, in-line styling and an alternative name can be given to the icon to gain better control of the styling.

3.4.5 LocalStorage use

The client has indicated that it is preferable to have some form of persistence from session to session. It has also been suggested that this should be done without using user registration, as this

[11] https://www.npmjs.com/package/i18next

is both intrusive to the user experience and introduces an unnecessary security risk.

Therefore, the design plan indicates that the application would make use of cookies. Although cookies have gotten a somewhat bad reputation and some people block them, they could still enhance the user experience for those users that allow them. Cookies would be used to store both the users' settings and presets. With a reasonable explanation for why cookies are used, it is expected that many users allow them to be placed.

During the development process, the decision was made to make use of localStorage instead of cookies. LocalStorage turns out to be easier to implement than cookies, not requiring extra libraries on top of JavaScript. Additionally, localStorage key/value pairs are never sent to the server. This slightly reduces overhead. In general, it has become standard practice to use localStorage instead of cookies when only the client side needs to read the stored values.

For the law, localStorage is similar to cookies. Thus, a cookie consent bar would still be needed if the application would be deployed to the public. Such a bar could be implemented with relative ease, but this has not been done yet.

Concretely, the following settings are made persistent by use of localStorage:

- The user's settings, such as language and colourblind modes
- The user's selected cultures
- The user's presets

3.5 Implementation

In this section nontrivial code of components will be explained. In that sense, it may be explaining the underlying structures, or the thought process behind it.

3.5.1 Forms

3.5.1.1 GeneralDropdown

The general dropdown is given only one required variable, the colour provider. Furthermore, it can receive a label, a list of options with both id and label, the selected value and a function to perform on select. Using this information, it invokes a dropdown from react-bootstrap over which the styling suiting to this project is applied. Additionally, each item in the list of the dropdown in itself is a Functional Component as it is required to flip the background colour and font colour on hover. This is achieved by updating the state.

3.5.1.2 GeneralSlider

The general slider is also only given a required colourProvider. To that can be provided, the label, the chosen value, the minimum and maximum value, and the action to perform on change. Then, it calls both a slider from antd, as well as an integer input from antd to better visualize the number. On top of that only positioning styling, and colour correction is done.

3.5.1.3 GeneralSwitch

This is the smallest of the forms in that it only requires to know whether it is checked or not, optionally a label and an onChange function can be provided. The only thing happening to this component is styling and colour correction on an antd switch. To create a correct background colour for all colour palettes a new background colour for selected switches is selected.

3.5.2 Icons

3.5.2.1 Icon

The Icon components prime function is to load in images from the public directory. Around that, a button is wrapped to get the onClick action that can be provided to the icon. To allow for optimal customizability, the developer can provide their own className and in line style to the component.

Whenever the icon fails to be loaded, it will fall back to the alt attribute. By default, this is an empty string, but another value can be provided - and is strongly recommended.

3.5.3 Modals

3.5.3.1 GeneralModal

The general modal is primarily their to create a highly customizable modal suited to the preference of this project. It overrides the antd Modal by providing a custom rendering function that erases the header and footer function so that the styling is as flexible as possible. Having done that, the only thing that is rendered is a popup screen with a close icon in the preferred style. The properties provided to this function contain the colour provider, whether the modal is visible, what happens on close, and what should be visible in the modal. The component is destroyed on close to force rerender of its content on reopening.

3.5.3.2 AddModal

This modal is one of two modals that provides a body to the general modal to render a functioning modal. The goal of it is to be able to mark which cultures to list or have selected, or to create a new preset. This is allowable by having a confirm function that provides the parent with the selected and listed cultures. In the case of creating a new preset - which is marked by a boolean - the name of the preset is also checked for uniqueness by going over the also provided taken preset names.

Moreover, internally this modal keeps its own list of selected and listed cultures. This decision is made so that the selected culture object of side menu can be overridden without it not allowing it due to it being the same reference. So, when a culture is clicked to be listed it will be listed in that internal copy, when it is marked as selected, it will be both marked listed and selected. Added to those features, the user is able to search through the cultures. This is done by going over the internally upkept cultures and checking whether the start of the label matches the provided string.

Rendering the body is done with the input bars on top, the search bar and preset naming - if needed. Below that are the indicators to mark which checkbox serves what purpose. The position of this is hard-coded as the scope only states to account for 1080p screens. There below, the cultures are placed one for one with their label and the check boxes to select or list the culture. Finally, two buttons are placed in the modal, one to confirm the change, or one to cancel the change.

3.5.3.3 PresetsModal

The sole purpose of this modal is to list the currently existing modals paired with each culture it lists. Additionally, whenever a culture will be selected by selecting that preset, it should be highlighted. Therefore, the only properties given to this function specifically are the available presets and the colour provider. The other properties are those required by the GeneralModal. Using the available presets, each is listed and paired with a cross icon to remove them. Finally, similar to the AddModal, two buttons are added to confirm or cancel all actions.

3.6 sidemenu

3.6.0.1 SideMenu

The only function of SideMenu is to pass the correct properties through to Cultures and Presets. However, it does keep track of an internal culture state to fake the aforementioned sense of responsiveness to the user. How that works is explained in Section 3.6.0.3

3.6.0.2 Header

This header is visible in the sidebar, dividing the two sections of cultures and presets. A separate component was created for this since it could be used multiple times and is easier to use. This header exists of four sub components: The title, save function, add function and settings function. The title is nothing exciting, it merely places the text to the left of the header.

The functions are optional. If no function was added to the component, the icon would not appear as well. It is possible to add any function to the header icons, with the appropriate icon accompanied by it.

3.6.0.3 Cultures

The cultures component renders the cultures section of the side menu. This does not only include the listed and selected cultures visible in the sidebar itself, but also the gear icon for the modal where the listed and selected cultures can be edited.

As mentioned before in Section 3.6.0.1, the cultures component makes use of the internal culture state provided by the SideMenu component to make the system seem more responsive. This is used to increase the responsiveness of the system by not updating the star display for every single change in cultures, but only after no change is detected for a set time.

In order to make this work, a debounce function is used. A debounce function is performed after a delay (which is provided as a paramater), but only if the function has not been called again within this delay. If it is called again, the delay resets to the given value.

With the use of this debounce function, it is possible to instantly update the list of cultures in the side menu, but only update the cultures provided to the Stars component after the debounce function has reached its delay. This way, the star display is not updated and rerendered entirely for every single change, increasing the responsiveness of the system.

3.6.0.4 Presets

The presets component is very similar to the cultures component described in Section 3.6.0.3, but it lists presets instead of cultures. These presets are also provided by the SideMenu component. When a preset is clicked, the list of cultures in the SideMenu component is cleared and then filled again with the cultures in the preset. These changes are made instantly and thus do not make use of a debounce function.

The presets component also contains two buttons in the header: one for adding a preset and one for managing presets. The buttons open a modal described in Sections 3.5.3.2 and 3.5.3.3 respectively.

3.7 Stars

3.7.0.1 Stars

The heart and soul of this project is contained in the Stars component. Within the constructor, it first determines the preference of the user; The gap size between edges, the width of the edges, the offset of the edge from the star, and sets those to the local storage if that information was unable to be retrieved. Additionally, it sets the list of the cultures it should display.

From thereon, the component will start to render into a preview screen, this screen displays the LoadingStars screen until the data have been fetched. So, fetching the data is started after the initial mount. Following that retrieval the component will start styling the nodes and edges relevant to the view of the user.

Styling the nodes and edges is specific to the view of the user. If the view is zoomed out, the node labels are replaced for a the zoomedOutNode component rendered as image. Doing this conversion is done by first converting the JSX to HTML, then converting it an SVG to finally convert it to an image. Additionally, it is matched with the cultures the star appears in.

Styling to the zoomed in view is a bit easier. In that case react-graph-vis handles most rendering, as no extra information such as the culture labels needs to be added. Therefore, each node is styled dependant on how the user prefers non-included nodes. Moreover, the opacity of each star is set adjusted to their magnitude. Following, the tool tip is added to each node in a close manner: Instead of converting it to an image, the ToolTip component is now provided in the JSON of the star as an HTML element.

After having styled the nodes, the edges will be styled. In this function it is determined whether the colours of each culture should be shown, or whether it should default to the font colour. In the latter case, all edges are combined by removing any duplicate from-to or to-from pairs. The former simply provides each edge with the colour associated to the culture of the edge.

Since react-graph-vis does not support displaying multiple edges alongside each other, a custom function is build for that to draw on the canvas manually. This is done by first ensuring that no from-to pair could exist as a to-from pair, i.e. a line from '0' to '1' could not exist as '1' to '0' at the same time. This needs to be ensured as otherwise the canvas would draw over its own edges. Hence, first an unmergedLines object is created in which each star is mapped paired with all other stars it is linked to via what colour.

Thereafter, the object is merged: This means that, for each key in unmergedLines it is first checked whether it already exists in the final object. If that is not the case, it is added as a new key with as value an object containing the destinations and colours. In that destinations array exists an array with in it objects with coordinates of the destination and the colours to draw. However, if the key already existed, the colours are appended to the destinations array of the destination-key.

With that ensured, the edges can be drawn on the canvas. First it is determined in which direction the line will be drawn as that will relate to the offset of the edge. Namely, if the line is going up left, or down right, the offset should be added to the start and subtracted from the destination. If it is going in any other direction, it should be the opposite way. Finding the offset is done by first determining the slope of the line, and from that angle determine the offset in both x- and y-direction using basic geometry.

Then, the lines will be drawn from left-to-right, each line first skipping half the preferred gap width, then drawing the edge, and then skipping the other half of the gap width. Doing it in that way, the lines will always be centered to the edge.

Next to filling the canvas with nodes and edges, extra events are added to make the system more intuitive. However, these functions are library specific and are explained in the docs of react-graph-vis.

3.7.0.2 LoadingStars

The purpose of this component is loading in the stars. During this time an animation will be played. For this animation we are using a library called react-content-loading[12].

That animation is created from a array of stars with a coordinate, which will be mapped over to create a circle, where the coordinates are multiplied by the width and height of the screen. That circle will then spin around.

A ComponentDidMount function is used to interact with the browser, so that the animation will immediately start once the website is visited.

3.7.0.3 ToolTip

This component shows up when hovered over any star. It serves the function of providing information to the user about the star and the constellation it is a part of. These tool tips are created when the star is initialized.

The data is generated by grabbing the constellations of the star and mapping over them, creating a separate section for each of them. In those sections the stars are mentioned that are in the same constellation together with which culture they belong to and the name of the culture. All of this data is gathered from the dataProvider class, which is given the star that is hovered over.

The colour of the lines that separate the title from the stars are picked from the colourProvider, so that they match the colour of the culture in the sidebar.

[12]https://www.npmjs.com/package/react-content-loader

3.7.0.4 ZoomedOutNode

This node is viewable when the user is zoomed out until the threshold is achieved. Only the most notable stars are shown in this version with each having coloured boxes that represent the cultures they belong to.

Those boxes are generated by a mapping function that goes through each of the cultures of the star and creating a small empty div that has a square shape and a background colour. That backgroundColour is picked from the palette that is given. The entire star with the boxes are then stored again in a div to be given to the view itself.

3.8 Pages

3.8.0.1 App

This is basically the container for the entire website. All of the other components are called from here. This component starts with assigning the default values to the settings if no previous settings have been set, for example the cultures and presets.

The other task it handles is storing all of the functions that are used deeper within the program. Each of those functions, that handle data such as the cultures, are defined so that they can be passed through to the other components.

Using such a container makes a clear hierarchy so that the program is build from blocks that each contain their very own specific purpose. This makes it easier to reuse those blocks at different places in your program.

3.8.0.2 Settings

In this component is the entire settings page defined. These settings are first gathered from the cookies, which are then used to set all the buttons in this page.

The first two settings are Language and Colour Palette. These are changeable by a dropdown menu which would then offer a few different options. This dropdown menu is in itself another component that was customly created for this purpose. When these values are changed, the entire page would then be updated to reflect the change immediately.

The next section contains three sliders which control the visual settings of the edges of the star view. Here it is possible to control the gap size between the lines, the size of those lines and the line offset. All of the values are stored locally in the settings file to be able to update them directly.

Then there are more general settings of the star view, namely dark mode, hide stars that do not exist in constellations, make those stars smaller, or make them transparent. These variables are again stored locally in the settings file.

Lastly, there are the import and export button. When the import button is clicked, a window will pop up. In this window the user can select the file they want to import, the requirement being that it is a JSON file in the correct format. Upon loading this document in, the file is checked on correctness. I then goes through each one of those settings and changes them in the program.

The export button is straightforward, as it simply downloads a file that is in a JSON format. To create this file, all of the settings are gathered from the localStorage. These files could then be exchanged with other users to import them themselves.

3.9 Utils

3.9.0.1 ColourProvider

This component provides colours to any elements that may be visible on the screen. The ColourProvider draws a list of palettes from palettes.js. This is done to make it easy to add new palettes, by adding a list of colours for the dark and bright modes of the new palette in the correct format. It also allows our settings page to load the same list from palettes.js.

There is some code in ColourProvider that could do with some explanation, which is the if statement in removeEdge. This statement checks whether the removed culture frees up a colour when no coloures were free before. In that case, the colour is assigned to a culture that did not have a colour yet, if such a culture exists.

3.9.0.2 DataProvider

Since this class contains many functions, only the nontrivial functions are explained in this section:

Requesting data from within any part of the system is done via the data provider. When a new DataProvider instance is created, the constructor will copy the edges over to an internal version. Additionally, it will figure which stars to include in the zoomed out version by counting the frequency of each star. If that exceeds the provided star significance, it will be included in the zoomed out version. The stars component uses a significance of 18 to have the highest 25% included.

Following the construction the user can request the stars for a provided x and y or the coordinates. Since it is desired for the user to be able to infinitely pan to any side, this must take into account wrapping of the data. Therefore, selecting the correct stars is done by first computing in which pan the user is. For example if the max x is 100 and the user's x-coordinate is 150, it would be in the second pan, $\lfloor 150/100 \rfloor$.

Then, for all stars, it will be determined which x, and y repetition would be in range for that pan. So, in the case of x-coordinate 150 and range 100, the star with x=30 would both be able to get selected in x=30, x=130, and x=230. This is achieved by expanding the pane into either a positive direction, to then expand it into a negative direction, whenever it is included in the range push it to the available coordinates. Note that, it may be that the user is on x=90 with a star on x=10 and a range of 50. Currently, that would fail as it breaks on fail. However, x=110 is in range of x=90, therefore the panning may fail to push once. With the selected list of coordinates available, the combination for all x and y coordinates is made, and each star in range is selected and returned with an id in the form of "{xPane}-{yPane}-{id}" and the realId as the original id, additionally, the x and y coordinates are adjusted.

Providing the edges of the view is done by first requesting the stars, to then group them based on their real id. This is done to prevent edges drawn across the entire canvas: Since the earth is spherical and the canvas is a square, it is unable to find the shortest edge by itself. For example, a star on x=90 and x=10 with a highest x=100 likely create an edge crossing the border of the square. However, that would not be supported by simply connecting the dots.

Therefore, the grouped stars are taking into account together based on their real id. Using that group, the data provider is requested to provide the shortest edge between two points - using basic geometry. With that information, the from and to of an edge will be put in an object bundled with the culture it belongs to, and whether it is self-referencing.

Next to the ability to provide nodes and edges, the system can provide the constellations that a star appears in for the provided cultures. This feature is primarily used for the tool tip and works as follows: The function iterates over all selected cultures and pushes the constellation to the result whenever either the from-star or to-star of the edge is the provided star.

4 Testing Overview

To make sure that the system works properly and is comfortable to use, a testing plan was set up. This included unit tests and hands-on tests.

4.1 Unit Tests

Whenever a new function was added, a unit test was also made. These tests were written before programming has started to guarantee the best quality of code. When it did not make sense to write a unit test for a specific function, the peer reviewer of the commit would ask for an explanation for the lack of a unit test, such that functions will constantly be tested when this was viable.

These tests were made for the ColourProvider and the DataProvider, since those were the only two suitable parts of the code that could be tested via unit testing. For the other parts of the project, a physical approach was used.

4.2 Hands-on Testing

To take care of the aforementioned problem, manual testing was taken place. When another feature was added, another project member was testing the addition. This project member would then report back to the coder with the feedback. These results are then incorporated directly into the program and sent back to the reviewer. When this cycle was finished the coder can move on to the next part.

When there was a finished version of the project, it was presented to other students that could use the program and give their own opinion about it. These interview were quite informal since it would speed up the process and not require any extra work. The opinions were then passed on to the other members of the project team and be discussed. After the discussion the problem was then solved.

During the poster presentation there was also a lot of feedback given. The screen we have received at the start could entirely be used for the demo because we created a physical poster next to it. From this feedback a few things were changed to improve the usability of the program.

5 Visual Design

In this section of the report, the visual design choices will be discussed that are used in the project.

5.1 The Main Screen

This screen (Figure 7) is the first thing that the user will see once it has visited the website. Because of this, it had to be understandable what was visible, so that the user could easily use the program. So it was chosen that there would only be two things presented on the main screen: The menu bar and the stars.

Firstly, there is the menu bar. The menu is again split in 2 sections: The cultures and the presets. Since these are 2 features that the user would probably want to use fairly often, they are initially visible. If the user inclines to collapse the menu bar, this is also possible by clicking on the arrow icon at the upper corner. This would then give the entire view to the stars. Later in this section, the cultures and presets will be discussed in more detail.

Secondly, there is the star view. In Figure 7, the program is currently in zoomed-in mode, where more information about the stars is given. The colour of the lines correspond with the culture in the sidebar, to make a clear connection for the user of which constellations belong to which cultures. If the user has added more than 12 cultures, the program resorts to making all the lines white (Figure 8), to prevent a colour disco on the screen, which would make it hard to look at.

If looked closely, stars with a lower opacity could be seen. These stars are not completely visible because the focus was laid on the constellations, and those stars do not belong to any constellation. So they remain visible but do not clutter the screen with more information than needed.

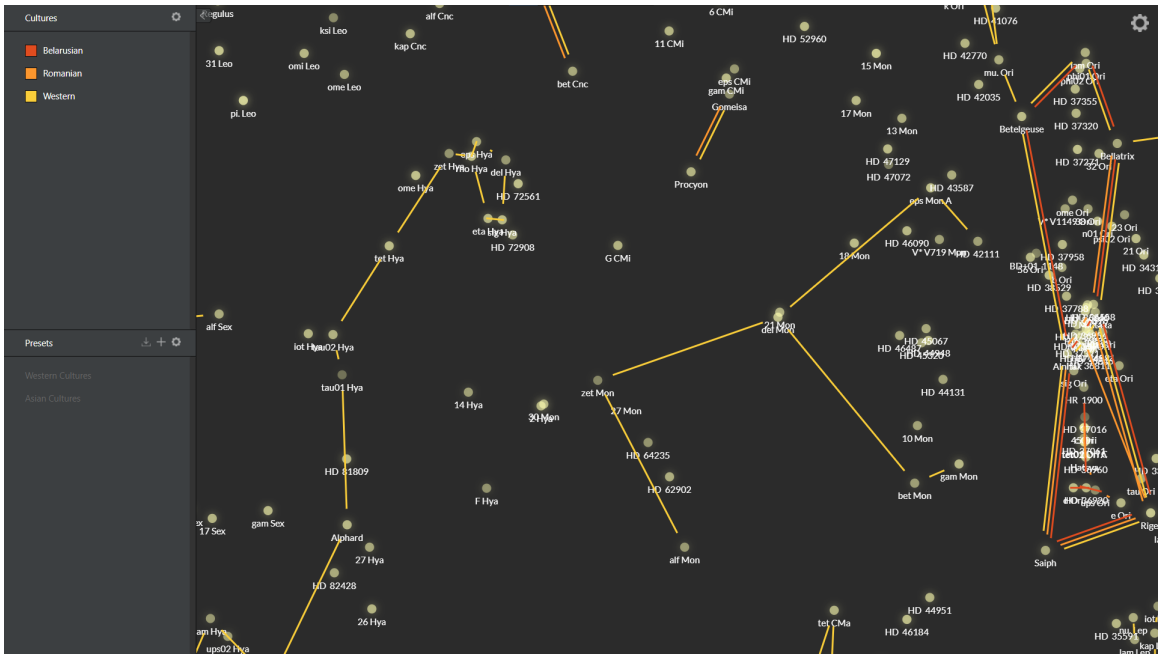


Figure 7: The main screen

To add some more information about the constellations themselves, an information screen was added (Figure 9). This screen would pop up once you hover over one of the stars in the constellation. On this screen, the user can see the rest of the stars in the constellation and the name of the constellation. Each culture is divided by a coloured line, again referring to the culture it belongs to. This is done for an easy way to compare the constellations of a culture, since it is neatly divided.

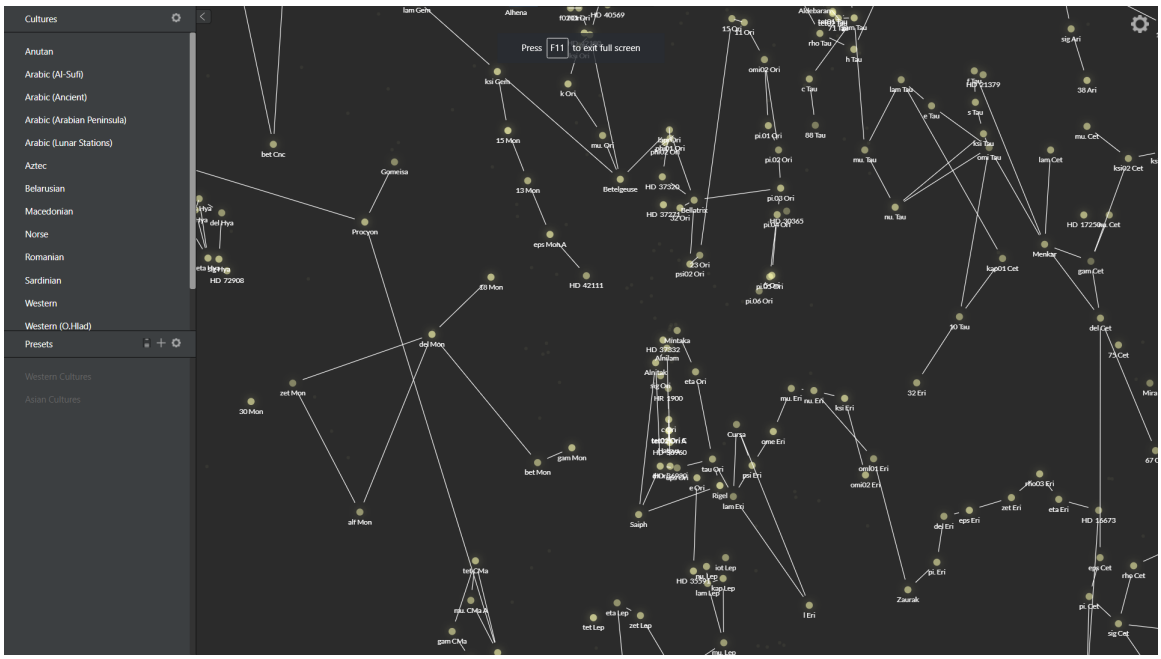


Figure 8: Too many cultures

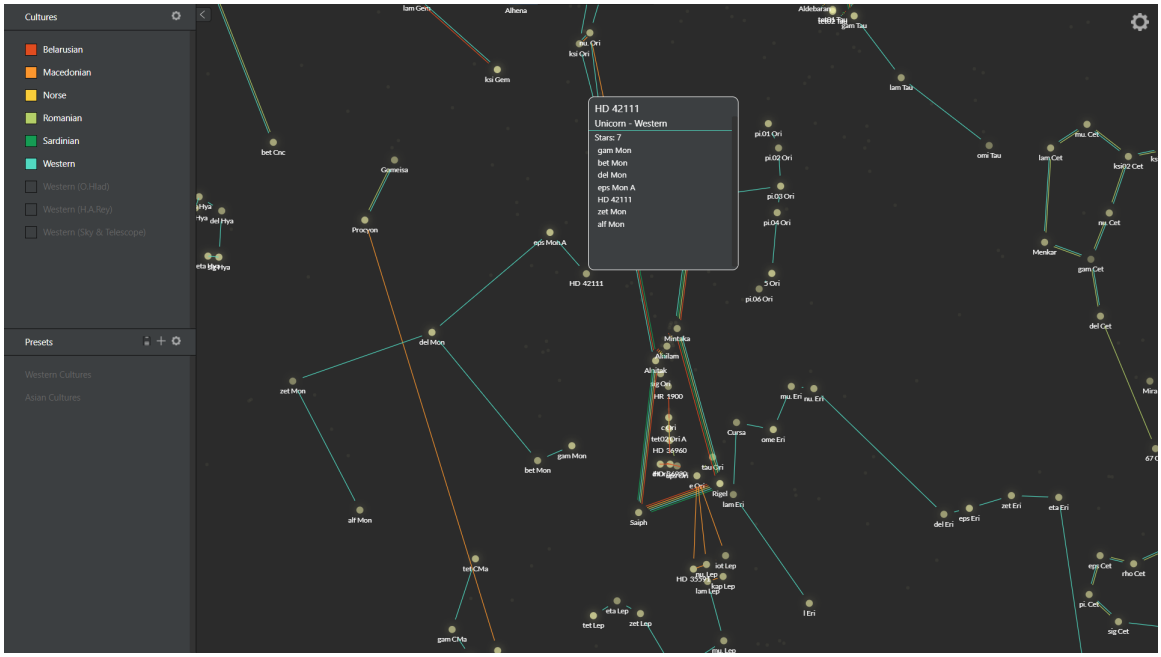


Figure 9: Information about the star

5.2 Zoomed-out view

For the zoomed-out view (Figure 10), a completely different design was thought of. Seeing as it would be a very busy screen if every star was drawn, it seemed a logical choice to only include the most important ones. To give the user a quick connection between the stars and the cultures, colourful blocks are added. ed. Each of those colours can be found in the cultures part of the sidebar as well, just like the edges in the zoomed-in view. So, in the example below, Caph, Alpheratz and Achernar all belong to the Anutan culture, but Cih does not.

5.3 Cultures

Coming back to the sidebar and specifically the cultures section, a list of current cultures can be seen. Each of these cultures can be turned on and off by clicking on them, this is made clear by hovering over the culture name and the text changes to a strike-through version. Once hovering over

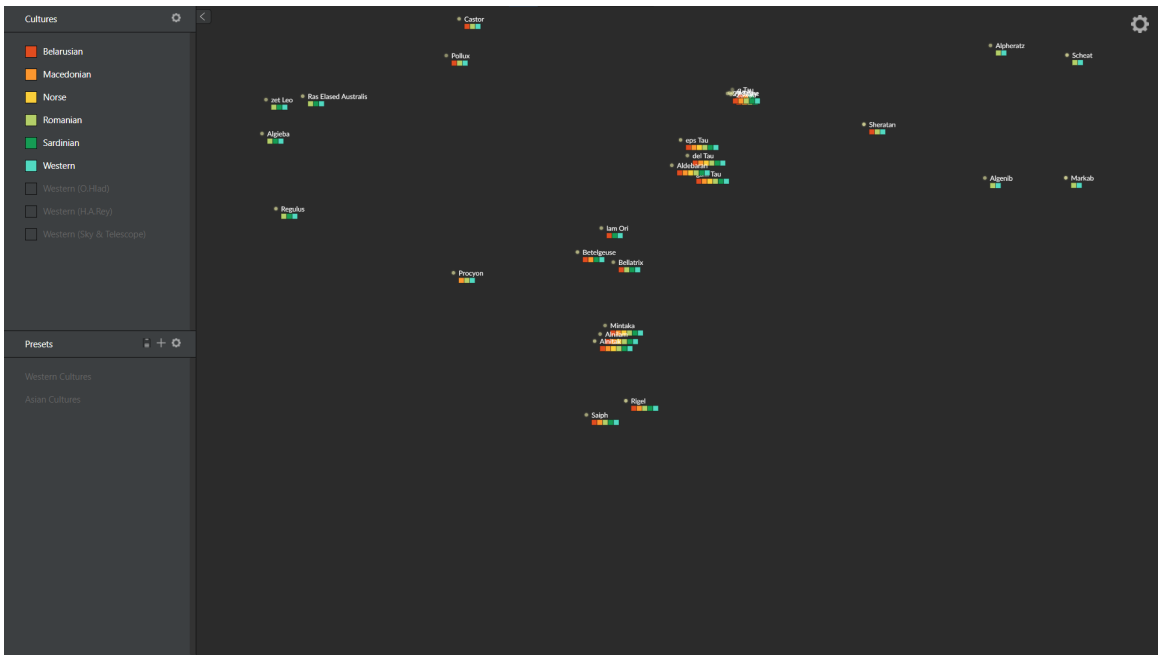


Figure 10: Zoomed-out view

the name, an "x" icon appears to the right side of the name. Clicking on this removes the culture. These designs were thought of to remove the use of an extra screen to quickly change your cultures.

If clicked on the cog icon of the cultures tab, a new modal would appear (Figure 11). In this modal it is possible to add or remove cultures from the sidebar. A search bar is added to the modal to help the user find their culture faster. There two options per culture: listed and selected. These options are provided so that the user does not have to immediately turn the cultures off if they desire to. If they only want to place them in the list, you can click on "Listed", if they want it to be activated as well the "Selected" box can be clicked. The design for this modal is kept simple, to not confuse the user.

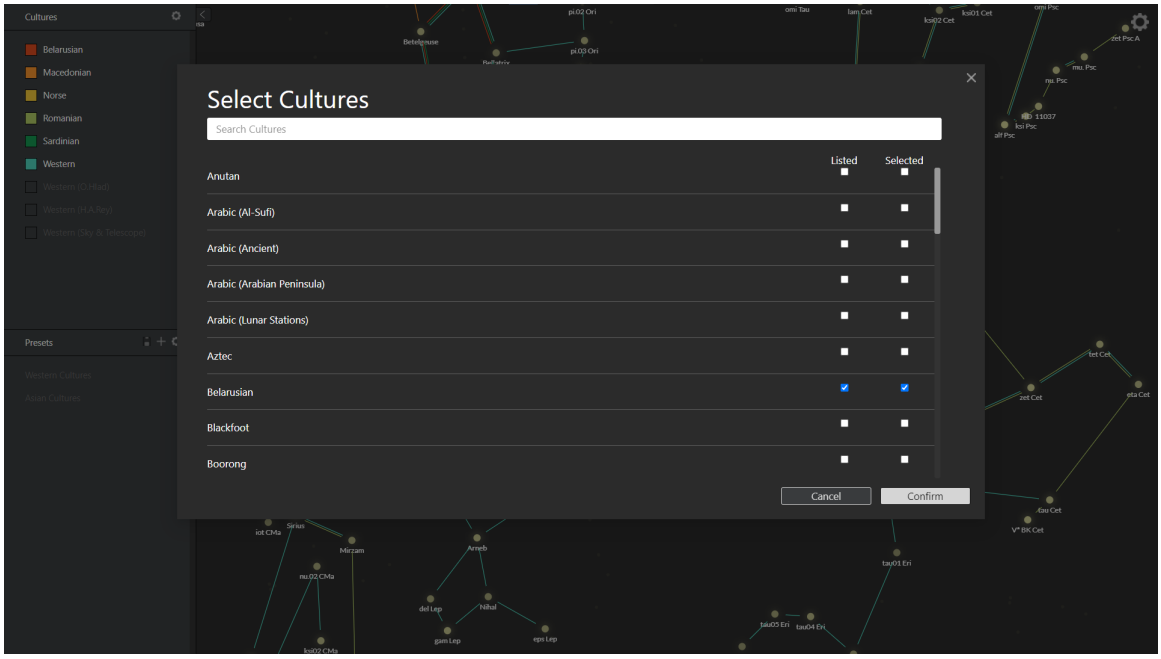


Figure 11: Cultures Modal

5.4 Presets

To make it easier for the user to view a selection of cultures that they often view together, they can make use of presets. When the user wants to add or use presets, the process is initiated from the side menu. For existing presets, a list is shown. The user can click on a preset to load the cultures from that preset.

When loading a preset, we opted to remove all existing cultures from the list of selected cultures, to not clutter the newly loaded preset. This is the behaviour that we find most natural to a user. We also remember which cultures were on screen in the preset, and which ones were only listed.

The user can easily remove a preset by clicking on the cross next to it. Further, there are three useful buttons above the list. The user can click on the save icon to save the current configuration of listed and selected cultures. The user can click on the plus icon to add a new preset from scratch, selecting which cultures are listed and selected (Figure 12). Lastly, the user can click the cog icon to load an overview of all existing presets and their configurations.

The modal for adding a preset (Figure 12) is essentially a list of cultures, that can be individually listed and selected in the preset. Furthermore, the modal includes two text input fields for naming the preset and for searching through cultures. To make the modal easy to read, lines were added under each preset to indicate which boxes belong to each preset. Furthermore, the modal was made more sensible by automatically selecting "listed" when "selected" is selected too.

The modal that shows an overview of the presets (Figure 13), is designed to provide the user with a way to quickly see all existing presets. This was done for two reasons. In the first place, the user can easily find the relevant preset that they want to select. Secondly, the user can easily clean up the list of presets when they see an obsolete preset, by clicking the cross icon.

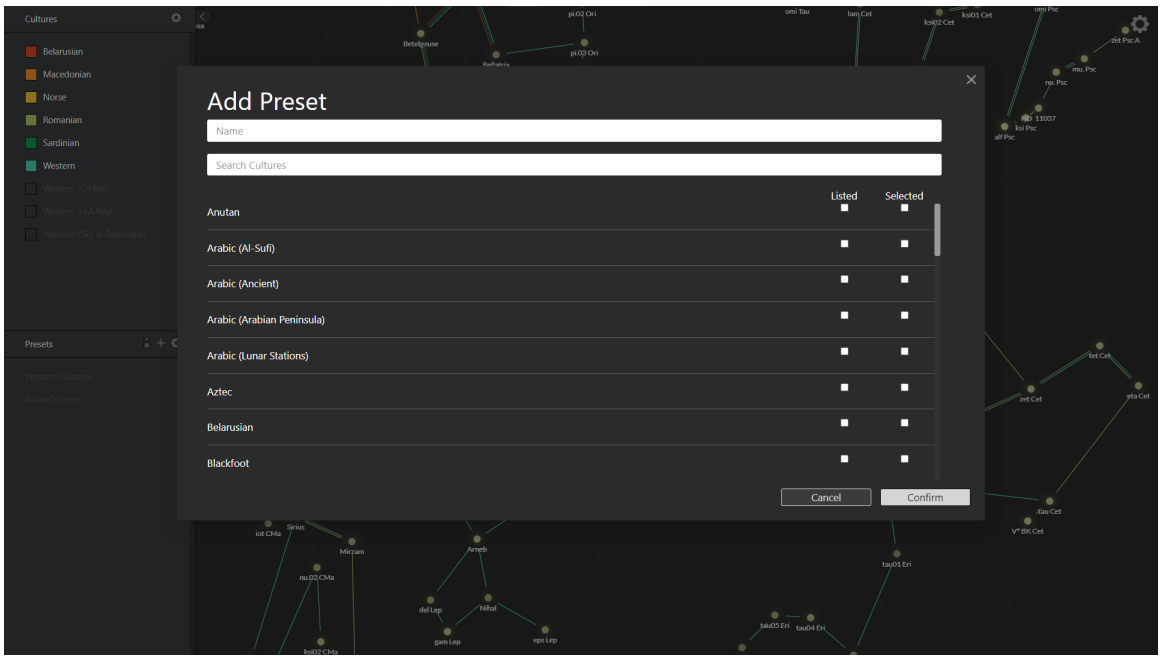


Figure 12: Modal for adding a preset

5.5 Settings

To add accessibility options to the project, a few settings have been added that the user could choose. These include language, which colours are used and if they want dark mode. The language setting is important for this project since the intention is that it should be usable by almost everyone. The colour mode is specifically made for people that cannot see colours well. These colours follow the standard for colour-blind people, and multiple versions are offered. A more in-depth

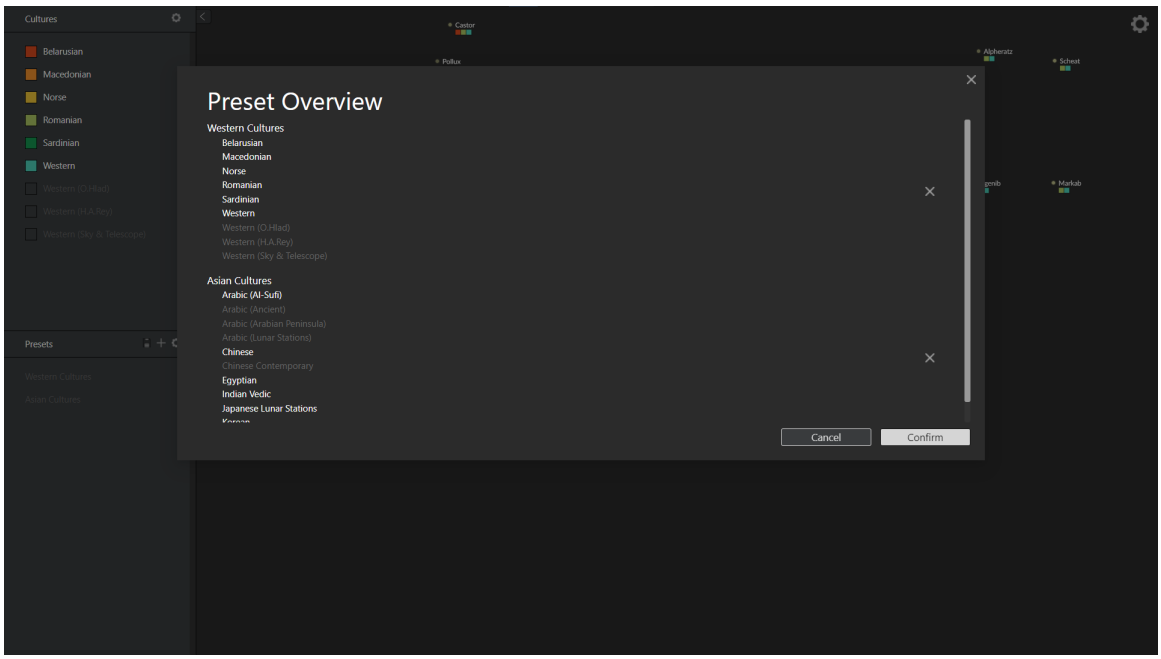


Figure 13: Preset overview

description can be read in the section below.

Dark mode is useful for the people that use this program in an environment that is not super bright, or if they user simply prefers the screen to be darker. As most programs today offer such a setting, it seemed useful to also add it here. Again the design is as clear as possible, since you do not want to confuse your user.

Finally, a set of sliders has been added to the settings. The sliders allow users to change the gap between edges, to change the width of the edges and to set a custom offset of the edges from the stars. This customizability is helpful for users with different screen sizes, users with difficulty seeing small prints, and simply to adjust for personal preferences.

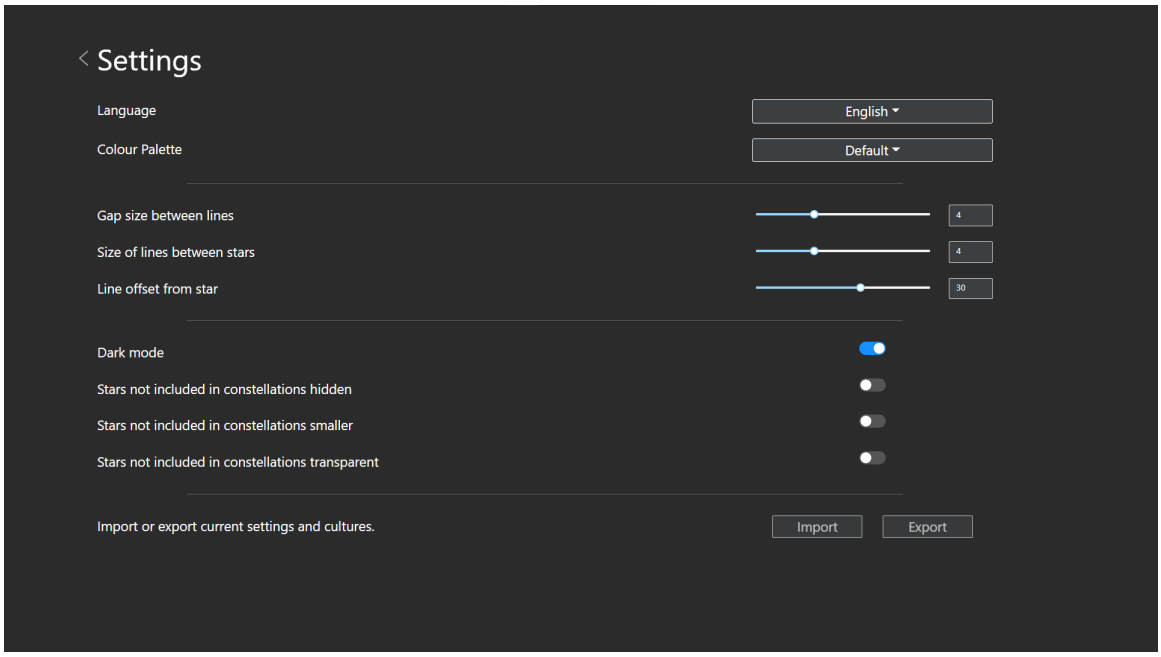


Figure 14: Settings page

5.6 Colourblind palettes

When providing accessibility for the (partly) colourblind, there are two options. The first is to provide a colour palette that works for everyone, in which case one palette suffices. However, this does imply that there are limitations as to which colours can be combined. Therefore, we opted to provide several colour palettes.

In each palette, we selected twelve different colours. The colours are selected such that colours that cannot be distinguished by people with a certain type of colour blindness are avoided in their respective palettes.[13] We made a default palette for people with regular vision, a palette for people with tritanopia, one for people with deuteranopia and a final palette for people with protanopia.

[13] http://mkweb.bcgsc.ca/colorblind/palettes.mhtml

6 Limitations And Bugs

Even though the main functionality of the program works, there are also some limitations which might hinder the user slightly when using the system. These limitations should however not be too apparent and should not cause much trouble to the user in any way.

6.1 Culture Selection

The first of these limitations can be found in the culture selection modal (see Figure 11). Here cultures can be listed and selected, and there is a search bar to make it easier for the user to find certain cultures. However, if the user has made some changes in this modal and they then enter some value into the search bar to filter the cultures, their changes are reset and nothing happens when confirm is clicked. The same goes for when changes are made to filtered cultures, and then the search bar is cleared. This most likely has to do with the fact that the cultures get rerendered when filtered.

6.2 Zoomed-in Star Display

The next limitation has to do with the star display. Since the star display is essentially a 'round' sky (as seen from the earth) plotted as a 2D map, stars located at the top or bottom of this map may appear further away from each other than they would in the night sky. A good example of this is the star Polaris, also known as the Pole Star, which is the brightest star that appears near the celestial poles. Since this star, on a 2D map, is displayed almost entirely at the top, it appears very far away from its neighbouring star (in the Ursa Minor constellation of the Western culture) although this is not the case in the night sky. See Figures 15 and 16 below for a visualisation.

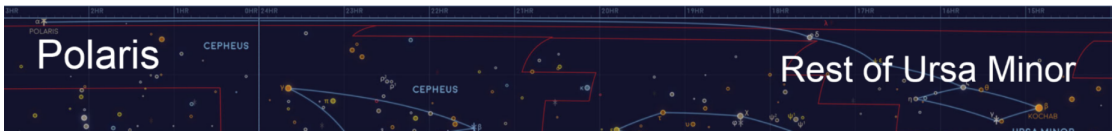


Figure 15: Polaris in the Ursa Minor constellation on a 2D map [14]

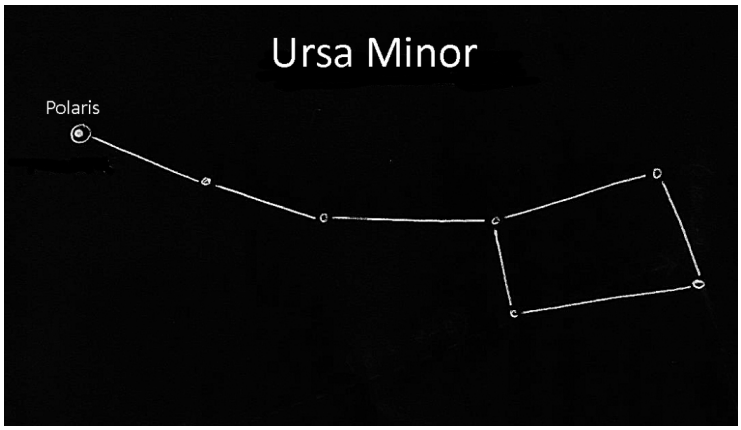


Figure 16: Polaris in the Ursa Minor constellation as seen in the night sky

There are two problems with this. The first of these is that this might give a distorted view to the user since they most likely would expect to see constellations the same way in the program as in the sky. This might make it harder to recognise constellations and compare them.

The second problem with this difference in view is the fact that the edge between stars that lie far apart on the 2D map is sometimes not drawn since the system only draws edges between stars that are in view for the sake of responsiveness. This might also give the user a distorted view of how a constellation looks.

6.3 Zoomed-out Star Display

Another small limitation is the fact that the zoomed-out star display does not update directly when selecting or deselecting a culture. Instead, it happens only when another action is performed (such as selecting another culture or removing one), or when the user makes a move in the star display. In the zoomed-in version (where constellations can be seen), the star display is updated directly.

6.4 Incorrect drawing of the edges

An issue that occurs in the program is the missing of edges between two stars where one of the two stars is not within the view that is rendered. As the program is first calculating which stars are within the range and then placing the edges, certain edges would be left out. Instead of this, a system with 3 layers should be made instead.

The first is a layer which would use the system as it is now created: only rendering the stars and their edges that are in range, and nothing else. The second would be a layer where the stars would be rendered so that it is possible to link to their middles. The third layer is the stars that are outside of the range, which would not be visible to the user.

7 Future Work

As we have reached the end of the module, the time to work on the project comes to a close. Sadly, it was not possible to add all of the functionality or features to the project as it was hoped. Because of this, the future work of the project will be discussed here, where we explain the points we wanted to work on.

7.1 Caching

The first point of improvement is the performance of the program. The data that the website uses currently is retrieved every time the website is visited. This is of course a huge loss of time, since it needs to be loaded in. An easy solution to this problem is caching, a technique where you store the data locally on the user's device, which the website could then retrieve. This is a much faster method than getting the data from the database that the program uses.

7.2 React functions

The second point is modernization. React has recently decided to move away from non-functional components, something that is used very often within the code of the project. Instead, React is focusing on functional components with hooks since they provide easier state and life cycle methods. These methods take care of the updating of the components. To keep up-to-date with the newest technology and improve the overall quality, it would be great to also implement this technology into our project.

7.3 Languages

The third point to improve the program would be accessibility. At the start of the project, this was an important point because the program should be usable by all of the people in our focus group, which includes everyone from the ages seven and up. To widen the range of people that can use the program, multiple languages could be added. Then the program could be understood by people that do not speak English well. Adding these translations can be done very easily: in the i18n.js file there is a JSON object that holds all of the translations, with each translation coming from a keyword that is inserted in the code. So by adding another language object to the file, you are making it more accessible.

7.4 Information about the cultures

The fourth point would be to add background information about the different cultures that are present within the program. Seeing that the cultures are playing the main role in the project, it would be nice if we could provide the user with information about those cultures. Then the user would learn more while also playing with the program and exploring the sky.

7.5 Option for joint edges

Coming back to the visual design of this project, enthusiasm was expressed by the supervisor about the way the program deals with too many cultures selected at once, namely the joining of the edges into grey lines. Thus it was decided that it would also be beneficial to the users' overview if it was an option in the settings menu, to allow it to be permanent regardless of how many cultures are selected.

7.6 Tutorial

Even though a manual is provided in this report, it would be a great help if first-time visitors of the website would be met with a tutorial. This tutorial would then explain the features and how to use each of them. It was tried to make the design as clear as possible for the user, but it would prevent any misunderstanding or confusion if such a tutorial would be added.

7.7 Edge highlighting

Mentioned in the achieved project requirement section was the problem of not being able to highlight the edges to show information about the constellation or cultures. Changing this to be

possible would require the switching to another framework or to create the function yourself. If more time was in this project, this would be a great point to work on.

7.8 Import button

Sadly, the complete functionality of the import button was not created. The button does import the language, cultures, presets and colour scheme, but lacks the other settings. The future work on this subject would entail adding those settings as well and adding an option for the user that is importing, which settings they actually want to copy and which they do not.

8 Risk Analysis

In the project proposal, a few risks were discussed that could be encountered at any point during the project. This section will look back at those risks and will reflect on them. The risks are divided into subsections, each covering a certain part of the project that are the biggest building blocks.

8.1 System Design

There were a few risks that were possible. The main risk was the restriction of the frameworks and techniques. Because frameworks do not offer complete freedom, most of the time, it is possible that it will hinder the progress when a certain technique does not function properly. This risk was a necessary one to take since building the entire project from scratch would take too long, and it would be a waste of time to redevelop something that has already been made.

Restrictions of a library were encountered during development of the project in several libraries. For example, limitations manifested in the cookies library that was intended to be used. The library was specifically tailored to React, which seemed like an invitation to use it with our project. As the developer who was to implement persistence was not experienced in React, it took quite some time before they realised that the module was not suitable for our project. As it turned out, the library made use of React hooks. React hooks, in turn, can only be used in React functions, while the project relied on Classes. Thus, the project switched to a different cookie library that was later refactored to use localStorage instead. However, this had taken a couple days to figure out.

When implementing the import button, a similar problem was encountered. React-bootstrap does not have a functionality to upload a file through a button. This problem was avoided by using the HTML input functionality. However, this button is not customizable, and our solution might not be the neatest of all. We currently styled our own custom button which is linked to the HTML input object. The HTML object is then hidden, effectively linking our own button to the functionality of the HTML input object.

Another one was the framework of the modals. The library that was used for this turned out to be not as customizable as first thought. Luckily this was easily fixed because the new framework that was picked for this did not differ much from the first one.

The second risk was the discontinuance of certain modules that would be used in the project. Because the program relies on modules, it is of great significance that those modules would be up-to-date or even usable. This problem was not found during the development but would be possible to arise when the program is a lot older.

8.2 Data Processing

The processing of data was quite clear of risks, although there remained one obvious one. If certain data would be to be updated, it would not necessarily be updated in the program, since most of the computing is done beforehand within the data conversion to save loading time. This risk is still possible since the data that is used has only been retrieved once, and is stored within the project. To combat this risk, it would require a direct link to the database. Because of the scope of this project, it was chosen to not combat this risk

8.3 Development Plan

When developing the product, it was of great importance that the development plan was followed. A great risk was the negligence of writing quality code, which would have made the process much more difficult than it needed to be. As multiple people were working on the project at the same time, each with different levels of skills, this risk had quickly become real. Luckily, one of the project members was proficient in the framework that we used, and kept maintaining the code.

8.4 Deployment Plan

Now that the project is finished, only one main risk remains. That is the risk of problems during the deployment, such as confusion when executing the plan or the failure of the program. To

prevent this from happening, a well explained plan is put in the manual section that the client can follow to deploy the project.

9 Manual

When the website is first loaded, the user is presented with a screen that displays stars and star constellations, along with a collapsible sidebar where cultures and presets can be selected and modified. In the top right of the star display screen, a settings icon is shown where the user can set their preferences. In the following subsections, it will be explained what each of these components exist of and how they can be used.

9.1 Side Menu

The side menu is made up of two parts, of which the first is the culture selection. The cultures displayed here are saved in the cookies, but when these are not stored, either because the user does not want to or because the page is loaded for the first time, the western cultures are shown by default.

In the cultures section of the side menu, the user can click on the name or icon of a culture to select or deselect it. When a culture is selected, it will show up in the star display which we will cover later in this section. When it is then deselected, it will disappear from the star display whilst still being visible in the side menu for quick access. To entirely remove a culture from both the side menu and the star display, the user can click the cross icon which is displayed to the right of the culture that is currently being hovered.

The other part of the sidebar is the presets selection. Here the presets are stored that are chosen by the user, which are also stored in the cookies if the user allows them to do so. By default, there are two presets available: 'Western Cultures' and 'Asian Cultures'. The user can view what these presets consist of by clicking on the gear icon in the top right of the presets selection in the side menu. Here, a list of cultures is displayed for each preset and there is a cross sign to delete a preset.

When the user clicks on a preset in the side menu, the preset is enabled, and the cultures stored in the preset are available from the cultures section of the side menu. From here, the cultures section works like normal again where cultures can be added, removed, selected, and deselected freely.

9.2 Star Display

The displaying of the stars is done in two parts as well: it consists of a zoomed-in version and a zoomed-out version. You can switch between these two modes by pinching on a touchpad or scrolling on the scroll wheel of a mouse.

When in the zoomed-out mode, only the most important stars are rendered. This was done to prevent cluttering of all the data that is available on such a large scale. Next to each of these stars, a colourful bar can be seen, depending on which cultures have been selected. Each colour represents a culture, of which the link between these can be found in the side menu. This makes it easy to see which stars are in the constellations of which cultures.

On the contrary, in the zoomed-in mode, a much more detailed view is presented to the user. Here the actual constellations are given, indicated by a coloured line between the stars that form it. Again, the colours of these lines correspond to the culture they belong to. When multiple cultures are using an edge between two stars, multiple lines are given.

An exception to this is when there are more than twelve cultures listed in the side menu. In this case, there is only one line shown between stars if there is an edge in any of the listed constellations. In this case, the colours are also removed from the culture display in the side menu. This is because of a design choice where it was decided that it was not possible for the project group to display more than 12 cultures at the same time while keeping it easy to distinguish them. On top of that, it seemed that there is no use in comparing that many cultures at once.

Finally, there is an information box when hovering over stars. This box lists each constellation along with its stars that this star is a part of. When the star has a constellation in too many cultures to fit within the box, the user can scroll up and down to see all constellations.

9.3 Settings Menu

The settings menu can be accessed by clicking on the gear icon in the top right of the main screen. To return to the main screen, simply click the arrow at the top. In the settings page, the user can select their preferred language using the dropdown menu. This will automatically, using the i18next library, update the language across the entire application. The second dropdown menu will let the user select the colour palette. This includes a regular palette, the three types of colour blindness, and a greyscale palette.

Below are four switches, each with their own function as the user prefers. The first is the dark mode switch. This will change the theme of the application to a dark theme. The other three are small preferences regarding stars that fall outside of the constellations. The first one will hide all stars that are not part of a constellation entirely, the second switch will make them appear smaller and the last switch will fade them according to their brightness.

Lastly, there is a functionality to export the currently selected cultures, presets and general settings. The export button will automatically generate a JSON file containing the necessary information, which is then stored on the device locally. With the import button next to it, a JSON file can be uploaded. This file is then verified for the correct information, and when a proper file is supplied, the settings, cultures and presets are applied in across the application. When the user does not supply a proper file, the file is discarded, and nothing is updated.

9.4 Setup

Setting up the project is done by simply building the Docker file and running it on the correct port. So if port 80 is desired it would work as follows:

```
$ docker build -t animated-sky .

$ docker run -p 80:80 animated-sky
```

10 Project organization

Reflection is a big part of learning from your mistakes and thinking what could be done better or what has been done well. This section will look at the different aspects of the reflection, such as: the group dynamic, the planning, individual process, client contact and supervisor contact.

10.1 Group Process

10.1.1 Group dynamic

The teamwork in the project group did not go smoothly all of the time. The communication between the project members was horrible at multiple points during the project. This caused a lot of delay, because people relied on the work done by other people.

The difference of work ethic between the project members was also quite large. As some members spent their time stressing over the work to be done and working on it deep into the night, other people were not at all concerned over the workload. A few discussions were had over this problem, hopefully preventing this problem from happening in the future.

10.2 Planning

There was a planning made in the beginning of the module which laid out clearly what was meant to be finished on what time. This planning was not adhered to by all members, also resulting in delay, wrong priorities and chaotic work overall. As some parts of the project were not completed on time, this would then push the rest of the work further as well, causing a lot of stress in the last week of the project.

10.3 Individual Process

10.3.1 Twan Boeve

At the start of the project, I was responsible for providing all the data needed to display stars and edges of the constellations of each culture. For the most part, this was not too difficult since the data for the constellations was provided by the client and could easily be accessed. However, calculating the x- and y-coordinates and retrieving the names of all stars was quite challenging.

For the rest of the project, I was responsible for the cultures section of the side menu and all of the modals used in the project (one for selecting cultures, one for adding a preset, and one for modifying presets). Due to having to rewrite parts of this code several times, either because the library used was unable to provide the right utilities or because the implementation deviated from the design in the mock-ups, this took longer than expected.

A final thing I want to mention is that the project team could not always rely on me being on time. We agreed that this would be okay if it was communicated beforehand with a good reason, but it would happen several times that I would appear hours later without letting the project team know anything about this. The team addressed their frustrations about this accordingly and I have been trying to improve this in the last few weeks of the project.

10.3.2 Pim Mulder

Due to personal circumstances during the final days of the Design Project, Pim was unable to write this piece himself. Thus, this part has been written to the best of our recollection.

From the beginning, Pim has consistently made large contributions to the project. In the beginning he was heavily involved with the mock-ups, eventually creating a higher quality version of them that closely resemble the result. He then went on to create a data provider that was used to create the star view, which he created as well, the zoomed-in and zoomed-out version. He helped the other project members with their problems and tried to figure it out along side them and refactoring code when it was written in a poorly fashion.

In addition to his technical contributions, Pim was the one to keep the rest of the group motivated. He would always work hard, and he often led the team meetings. Besides, he gave

useful feedback when it was needed. The English level of the project was lifted as well because of his thoroughness. All in all, Pim has been the main contributor to the project.

10.3.3 Patrick van Oerle

During the start of the project, I was unfortunately absent due to illness. The main problem was that I did not communicate this with my group, to whom it appeared that I was mysteriously missing. This has been discussed and I think I have improved on that a lot during the rest of the project. During the project, I was responsible for the settings page. This included the translations, implementation of the dark mode and themes, and the functionality to import and export settings as a JSON file. Furthermore, I have been working on the reports and have designed the poster for the poster presentation.

10.3.4 Wout Velthuis

At the beginning I was taking care of the documents, putting a lot of work in the project proposal and eventually converting the entire report into a LaTeX version. The translation was also figured out and provided by me. After finishing this, I went on to create the presets part of the project. This took me some time as I was not knowledgeable with React in the beginning. When working on this, I was tasked with creating a general modal that each of the more specific modals could be used, essentially making a template for the rest to work on, this was done together with Twan. I also helped and discussed other parts of the project, such as the designs, with the other project members.

Together with Jelle, I was mainly responsible for the ethics report that was handed in. It took quite some time to create a report on the ethical problems our project could pose, as we did not have a lot to do with user data or such. In the end, we thought of a few ideas that could be interesting to talk about.

I took the lead on creating this final report, as other people were still busy with their parts. I created a skeleton for the structure and assigned tasks to the other project members.

10.3.5 Jelle van den Wijngaard

In the initial phase of the project, my main contributions were in the form of good design ideas. For example, I came up with the coloured boxes used to indicate cultures in the zoomed-out view. I also contributed to the initial sketched mock-ups.

Functionality-wise, I was responsible for everything related to persistence/localStorage and I contributed to the import/export functionality of the app. Additionally, I developed the colour provider, which is used to serve the several elements that need colours.

Next to my technical contributions, I tried to help the team by discussing ideas and design choices. This often resulted in better design choices than would have been made on one's own. I also tried to keep track of deadlines, to lift some of the organisational burden off the rest of the project group. Similarly, I was responsible for maintaining contact with our supervisor and client, who were the same person.

10.4 Contact

For this project, the client and project supervisor were one and the same person. This meant that during meetings, they had two tasks to fulfill at the same time. This had both its advantages and its disadvantages, but first the role as a client and as a supervisor will be discussed separately.

10.4.1 Client

In our opinion, the role of the client often appeared to take a second place in our meetings. For example, when we tried to gather requirements at the beginning of our project, we were pointed to the initial description, a broad idea rather than a worked-out concept. That is fine for a project, and it was fun to have so much freedom, but it is not what we would expect of a client. As a client, valuable feedback came at the end of the trajectory when we gave a live demo. This feedback

could have been given from the mock-ups, which would have allowed us to incorporate it earlier on.

However, the above was not only the fault of the client. As a project group, we failed to provide early versions of the system because it was not functioning until the last two weeks of the project. This meant that we could not send the client any screenshots of the project, apart from the mock-ups and early designs.

10.4.2 Supervisor

We did think the role as a project supervisor was fulfilled very well, although the communication should have been better. This was mostly the project group's fault, as it would occur more than once that several group members would not be present on time or that the group would be slightly late for meetings.

Still, useful information was gained during the meetings. The supervisor was often able to provide feedback on subjects such as our planning and priorities, and would not hesitate to let us know if we were approaching tasks incorrectly. Although mistakes were often communicated very directly and could have been communicated in a friendlier manner at times, this did help us a lot in knowing what could have been done better.

10.4.3 Combination of Client and Supervisor

As mentioned before, there are both advantages and disadvantages of the client and supervisor being the same person. The main advantage is that instead of having to meet with both people separately, we would only have to meet with one person, making the meetings more efficient.

However, we would have liked it more if it had been two different people. The main reason for this is that it appeared difficult for them to provide feedback for both roles, as in our view we mostly received feedback from the supervisor only. This did not make it easier for us to design the system, since we often would not know what exactly the client was expecting in different parts of the system.

11 Presentation

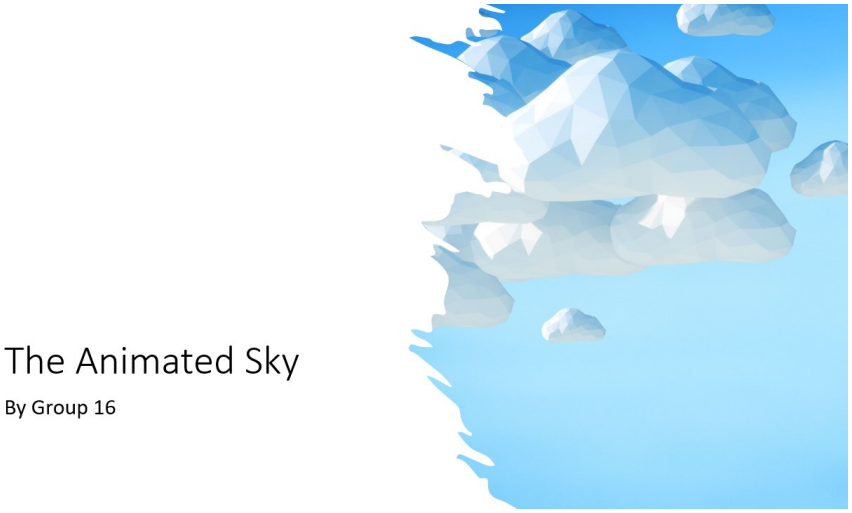


Figure 17: Slide 1: Front page

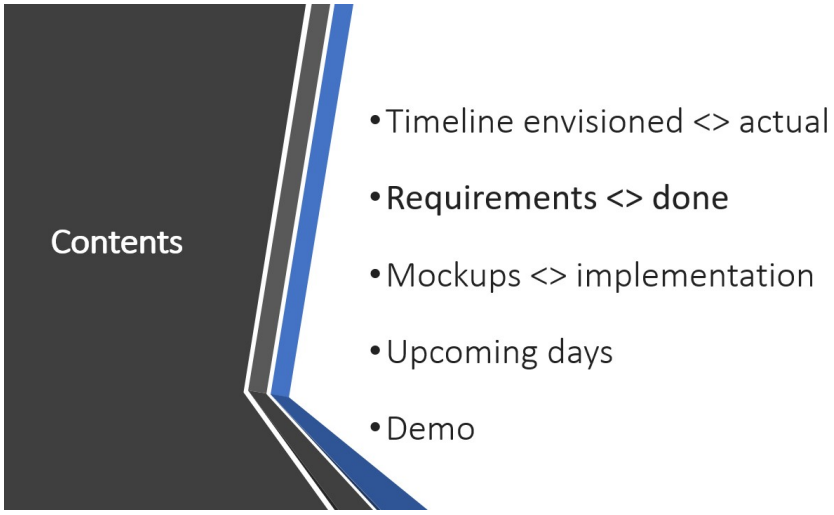


Figure 18: Slide 2: Contents

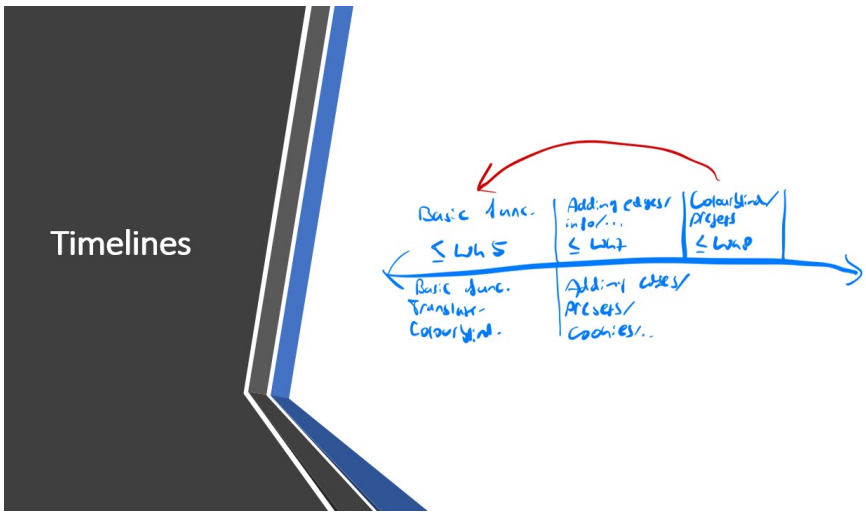
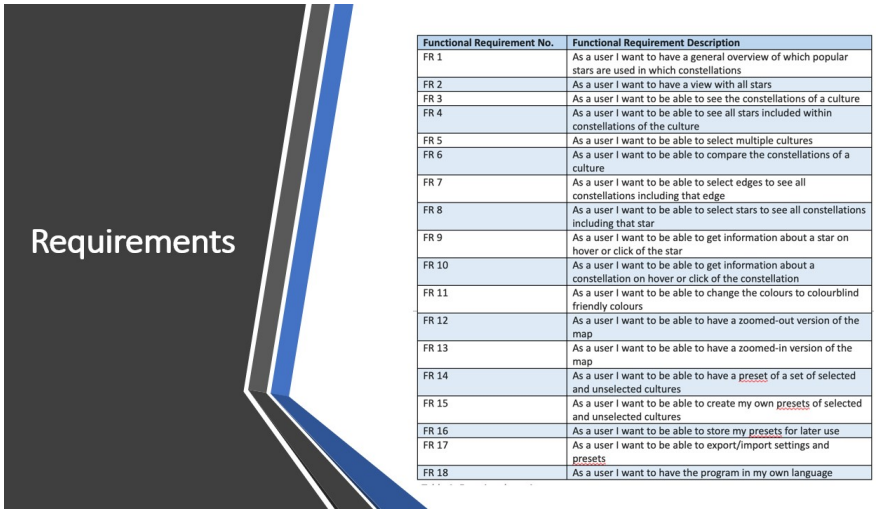


Figure 19: Slide 3: Timelines



Functional Requirement No.	Functional Requirement Description
FR 1	As a user I want to have a general overview of which popular stars are used in which constellations
FR 2	As a user I want to have a view with all stars
FR 3	As a user I want to be able to see the constellations of a culture
FR 4	As a user I want to be able to see all stars included within constellations of the culture
FR 5	As a user I want to be able to select multiple cultures
FR 6	As a user I want to be able to compare the constellations of a culture
FR 7	As a user I want to be able to select edges to see all constellations including that edge
FR 8	As a user I want to be able to select stars to see all constellations including that star
FR 9	As a user I want to be able to get information about a star on hover or click of the star
FR 10	As a user I want to be able to get information about a constellation on hover or click of the constellation
FR 11	As a user I want to be able to change the colours to colourblind friendly colours
FR 12	As a user I want to be able to have a zoomed-out version of the map
FR 13	As a user I want to be able to have a zoomed-in version of the map
FR 14	As a user I want to be able to have a preset of a set of selected and unselected cultures
FR 15	As a user I want to be able to create my own presets of selected and unselected cultures
FR 16	As a user I want to be able to store my presets for later use
FR 17	As a user I want to be able to export/import settings and presets
FR 18	As a user I want to have the program in my own language

Figure 20: Slide 4: Requirements

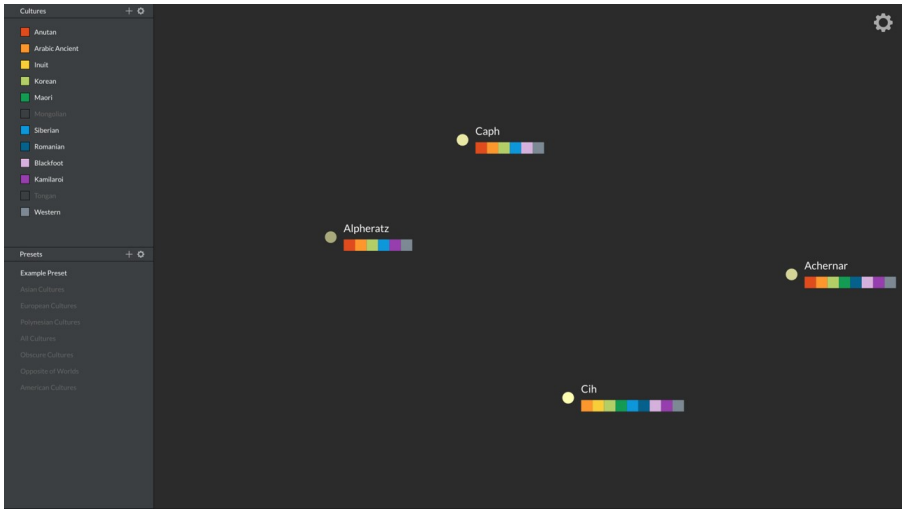


Figure 21: Slide 5: Zoomed-out view (1)

Figure 22: Slide 6: Zoomed-out view (2)

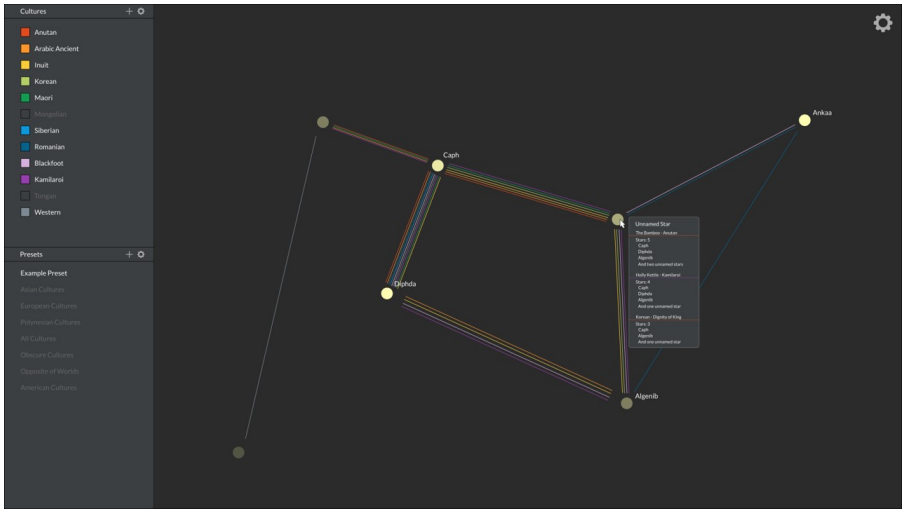


Figure 23: Slide 7: Star hover(1)

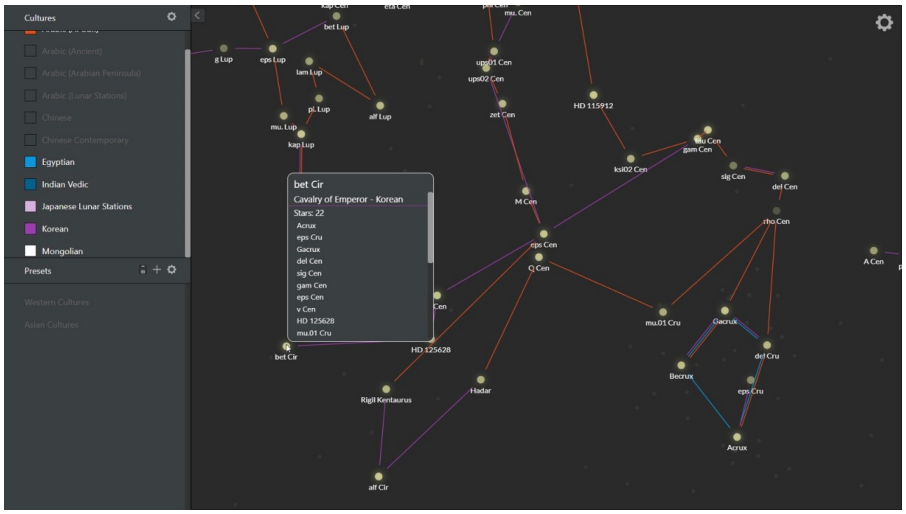


Figure 24: Slide 8: Star hover(2)

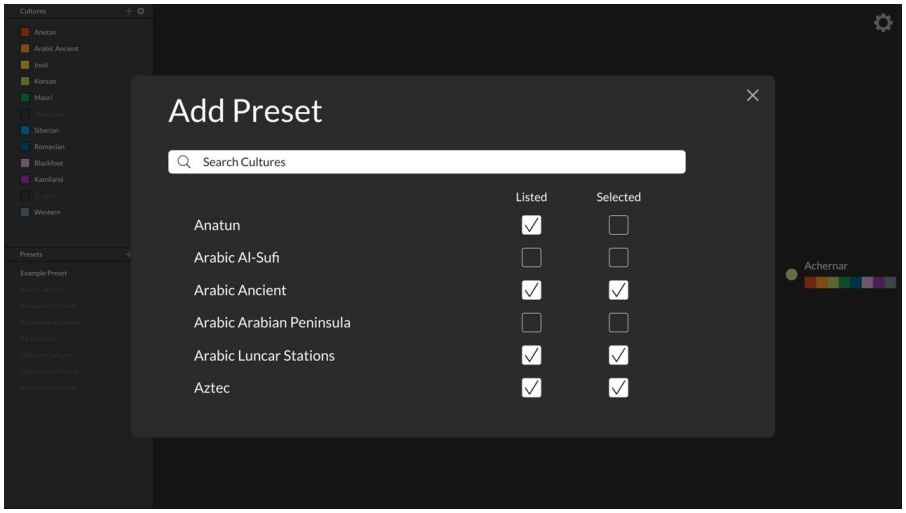


Figure 25: Slide 9: Presets(1)

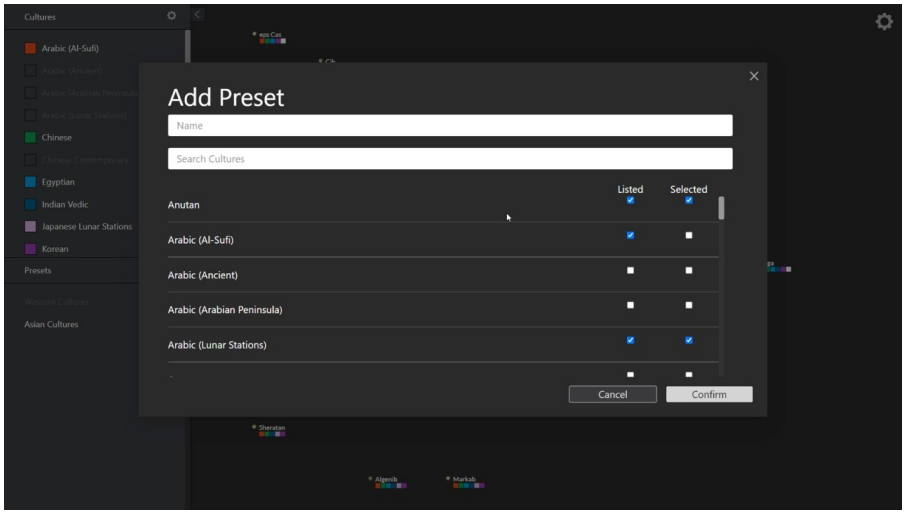


Figure 26: Slide 10: Presets(2)

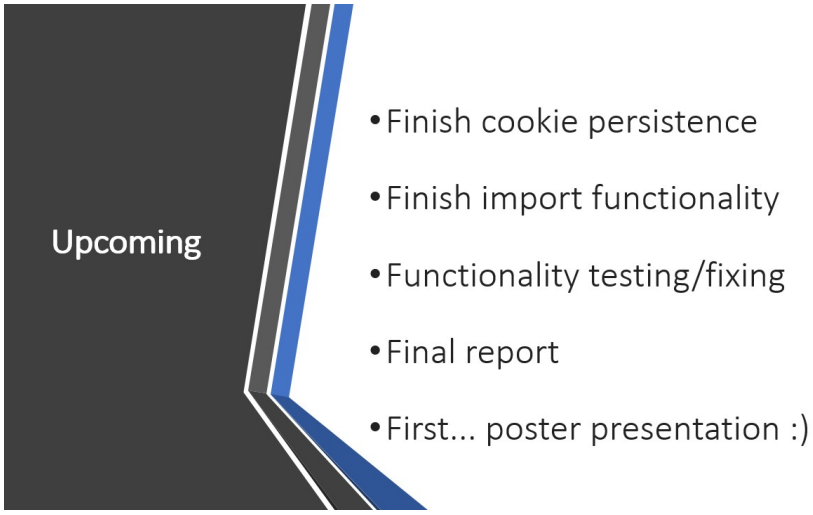


Figure 27: Slide 11: Upcoming tasks

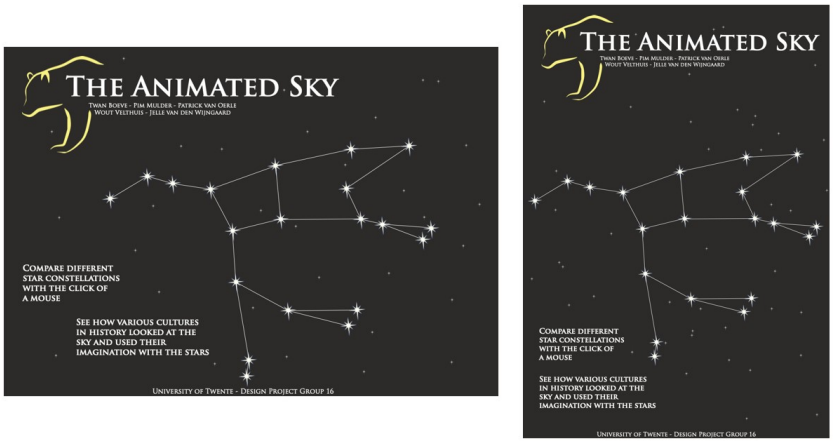


Figure 28: Slide 12: Posters

Figure 29: Slide 13: Demonstration

12 Poster

For our poster, we decided to go for a quite minimalistic design. This is to attract the attention of interested students and teachers, after which we can tell in more detail what we have done and what our product is. During the poster presentation, we decided to put our product on the CTOUCH screen that was provided to us, with besides an old-fashioned physical poster. We think this worked well, although Arend Rensink thought it was cheating in the best poster competition. A landscape version was also made, but we decided to have a live demonstration on the screen instead of our poster.

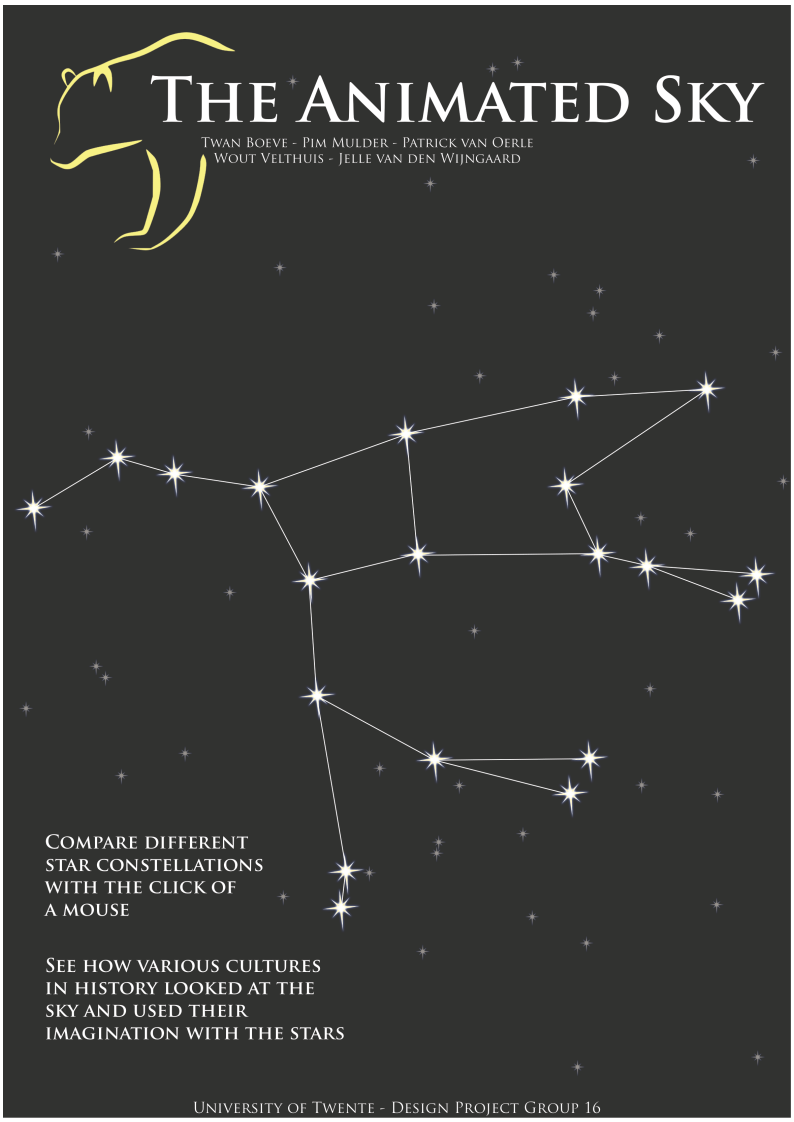


Figure 30: Poster used during the poster presentation

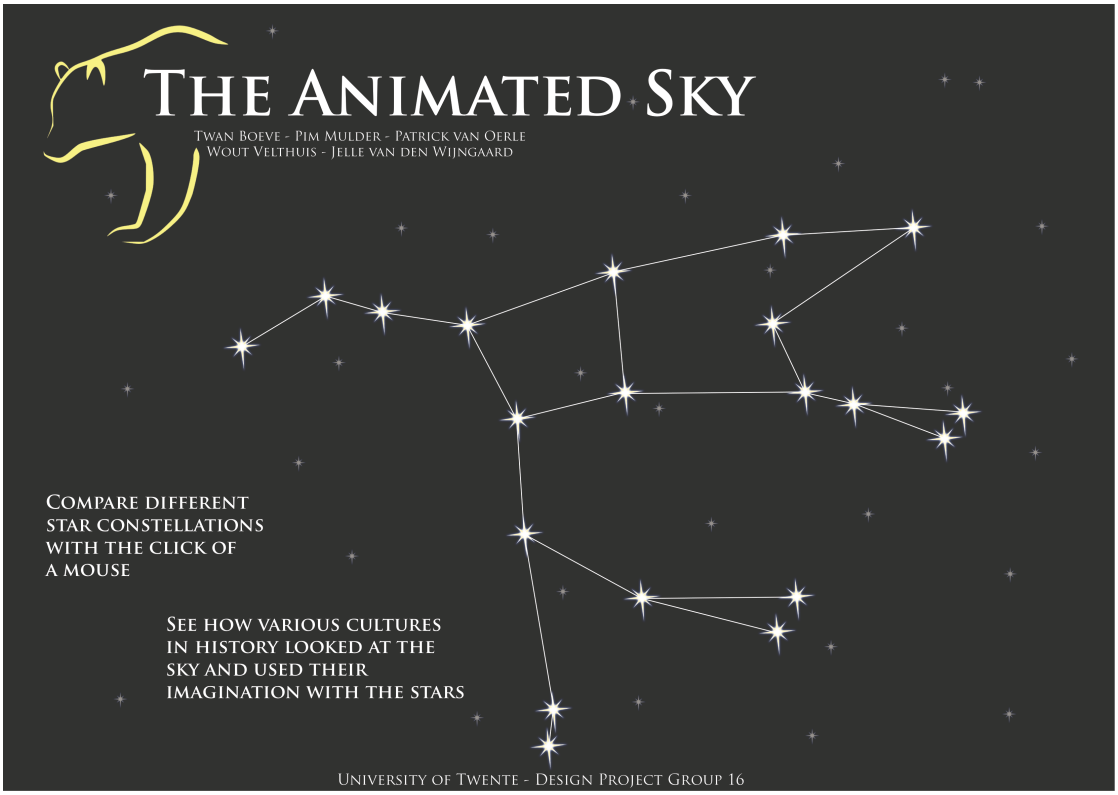


Figure 31: Horizontal version of the poster