SW Design project design document

Project Idea

Our project idea is a desktop application you can use to search astronomical events happening near you. The application contains a map and calendar to select desired coordinates and time range, and then based on those parameters forms API requests to various services to fetch information on various happenings in the sky. These events can be things like the International Space Station flyover, full moon, meteor flying over you, solar eclipse or more.

The idea was produced using ChatGPT with the following prompt: “Give me an idea for a JavaFX app that has a map with location and time selectors, and uses those to display data from various sky related events happening near given parameters. The information is fetched from external APIs.”

UI Prototype & Software architecture

Currently

Map architecture:

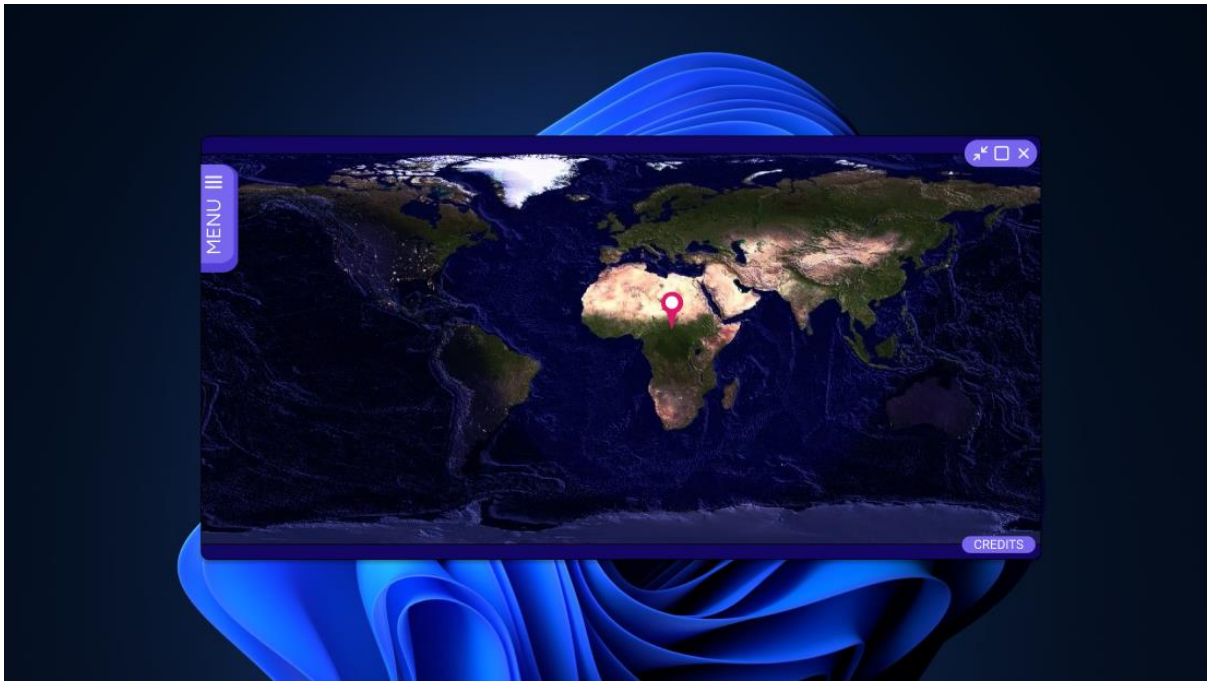
Since there were no good map libraries available / working at this point in time, our map uses an SVG image, and we draw our markers as well as calculate coordinates by converting the pixel coordinates to real coordinates and back.

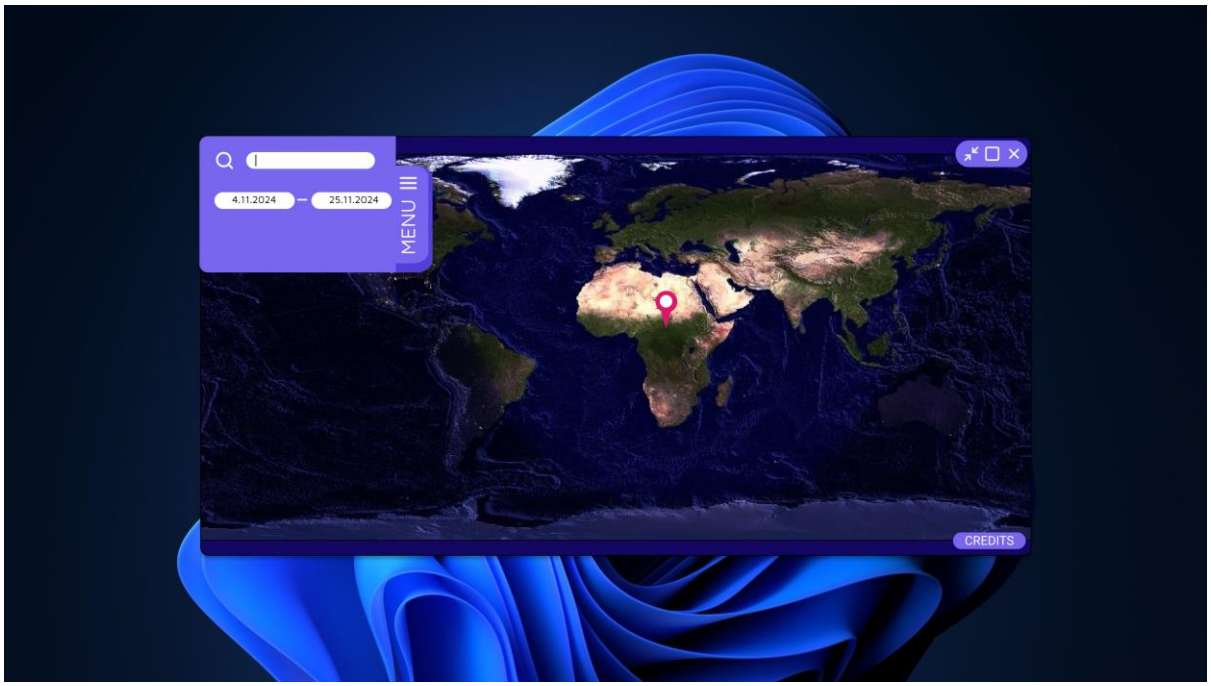
First we have SVGMap class, which takes in the path to the SVG file and the projection it uses in the constructor. Then we have SVGLoader which uses Apache Batik –library to convert and display the SVG file in our JavaFX application, since SVGs are not supported by JavaFX out of the box.

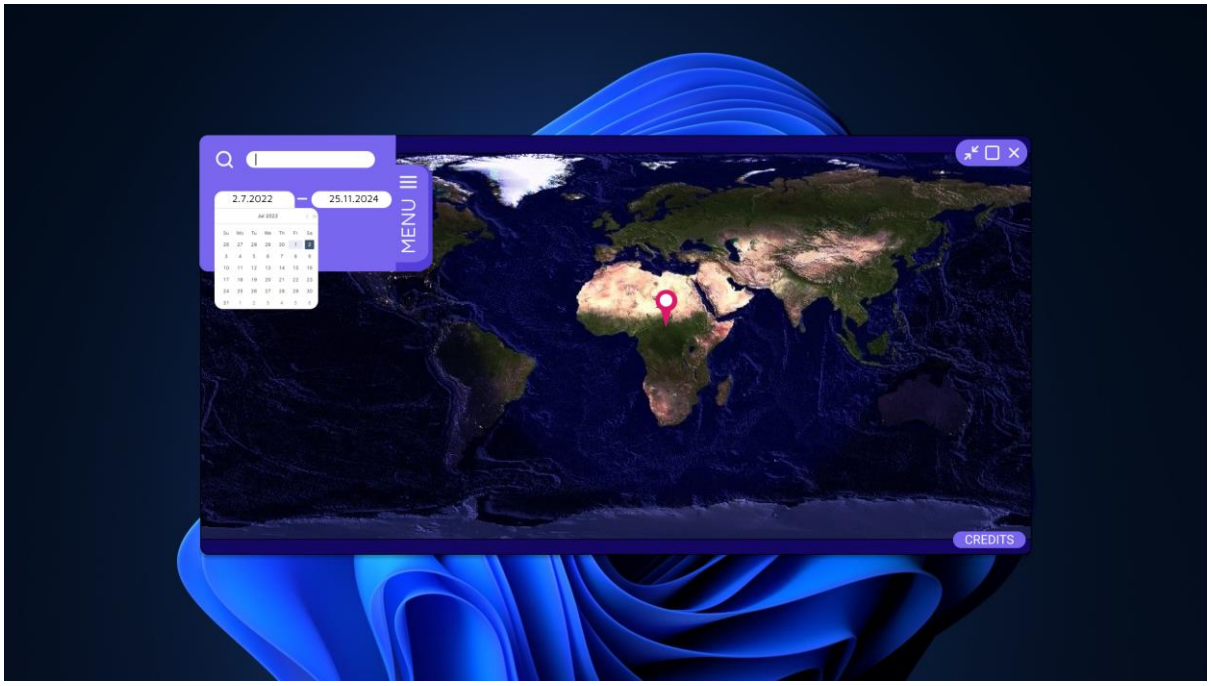
Then we draw markers over the map, and based on the pixel coordinates Projector-interface implementing class converts the relative pixel coordinates into real coordinates. For example, if we wanted to change our Equirectangular map to Mercator map, all code except the EquirectangularProjector are completely reusable.

Preliminary diagram for the application architecture:

https://tuni-my.sharepoint.com/personal/teemu_ranne_tuni_fi/_layouts/15/onedrive.aspx?e=5%3A37a1aa982f984e2e8c81c44a3808ebb8&sharingv2=true&fromShare=true&at=9&CT=1726823942915&OR=OWA%2DNT%2DMail&CID=3e8ccd09%2Dd025%2De93d%2D11b8%2D65c2d0b924f4&cidOR=Client&id=%2Fpersonal%2Fteemu%5Franne%5Ftuni%5Ffi%2FDocuments%2FCOMPSE110%20Ryhm%C3%A4ty%C3%B6&FolderCTID=0x012000949387941A9FF445B5D10AC8DD6443BA&view=0


MENU
CREDITS


4.11.2024
25.11.2024
MENU
CREDITS


2.7.2022
25.11.2024
MENU
CREDITS


Hervanta
4.11.2024
25.11.2024
22:00
10.11.2024
EVENT TITLE
5:30
13.11.2024
EVENT TITLE
1:30
15.11.2024
MENU
CREDITS

Figma: https://www.figma.com/design/HTmK6DYGYL2gKKqHZ9VqiS/Astronomy-Map?node-id=0-1&t=6PNN8vP4upkbBMri-1

APIs to be Used

Astronomy API

1. Overview

The Astronomy API provides data on celestial bodies, including their positions and significant events (e.g., eclipses). It allows users to query data based on the observer's location and time for specified celestial bodies.

2. Endpoints

2.1 Get All Bodies Positions

- **URL**:

  `GET https://api.astronomyapi.com/api/v2/bodies/positions`

- **Description**: Returns the positions of all celestial bodies (Sun, Moon, planets, etc.) for a given location and time range.
- **Query Parameters**:
  - longitude (string): Longitude of the observer.
  - elevation (string): Elevation in meters.
  - from_date (string): Start date (YYYY-MM-DD).
  - to_date (string): End date (YYYY-MM-DD).
  - time (string): Observer's local time (HH:MM:SS).
  - output (optional): Format (table or rows). Default: table.

2.2 Get Body Positions

- **URL**:

  `GET https://api.astronomyapi.com/api/v2/bodies/positions/:body`

- **Description**: Fetches the positional data for a specific celestial body (e.g., Sun, Moon) for the given date range from the observer's location.
- **Path Parameter**:
  - `body` (string): ID of the celestial body (e.g., "sun", "moon").
- **Query Parameters**:
  - `latitude` (string): Latitude of the observer.
  - `longitude` (string): Longitude of the observer.
  - `elevation` (string): Elevation in meters.
  - `from_date` (string): Start date (`YYYY-MM-DD`).
  - `to_date` (string): End date (`YYYY-MM-DD`).

- `time` (string): Observer's local time (`HH:MM:SS`).
- `output` (optional): Format (`table` or `rows`). Default: `table`.

2.3 Get Events for a Body

- **URL**:

  `GET https://api.astronomyapi.com/api/v2/bodies/events/:body`

- **Description**: Returns celestial events (e.g., solar eclipses, moon phases) for the given body (Sun or Moon) over a specified date range.
- **Path Parameter**:
  - `body` (string): ID of the celestial body (currently only "sun" or "moon" supported).
- **Query Parameters**:
  - `latitude` (string): Latitude of the observer.
  - `longitude` (string): Longitude of the observer.
  - `elevation` (string): Elevation in meters.
  - `from_date` (string): Start date.
  - `to_date` (string): End date.
  - `time` (string): Observer's local time.
  - `output` (optional): Format (`table` or `rows`). Default: `table`.

2.4 Generate Star Chart

- **URL**:

  POST https://api.astronomyapi.com/api/v2/studio/star-chart

- **Description**: Generates a star chart image and returns the image URL.
- **Headers**:
  - Authorization (string): Basic authentication token.
- **Request Body**:
  - style (optional, string): Style of the map to be generated (defaults to the API's default style if not provided).
  - observer (object): Includes the observer's `latitude`, `longitude`, and `date`.
  - view (object): Defines the type and parameters for the rendered image.

    Constellation View
    - **Description**: Generates an image of a constellation based on the 3-letter constellation ID.
    - **View Parameters**:
      - constellation: 3 letter ID
  - **Area View**
  - **Description**: Generates an image of an area in the sky. Optional zoom is supported.

- **View Parameters**:
  - `parameters`:
    - `position`: equatorial coordinates (Right Ascension and Declination)
    - `zoom` (optional): Image scale.

Where the ISS at? API

Overview

The WTIA REST API provides data on the current, past, or future position of the ISS, timezone information for specific coordinates, and TLE data. No authentication is required, but requests are limited to roughly 1 per second.

Responses

- **Default Format**: JSON
- **Success**: 2XX response codes
- **Errors**: Non-2XX codes with error details

Endpoints

Satellite Position

- **URL**: `https://api.wheretheiss.at/v1/satellites/25544`
- **Description**: Returns the ISS's position, velocity, visibility, and other details at the current time or a specified timestamp.
- **Parameters**:
- `units` (optional): Miles or kilometers. Default: kilometers.
- `timestamp` (optional): Unix timestamp to specify the time

Satellite Positions for Timestamps

- **URL**: `https://api.wheretheiss.at/v1/satellites/25544/positions`
- **Description**: Returns the ISS's position and velocity for up to 10 specified timestamps.
- **Parameters**:
- `timestamps`: Comma-delimited list of timestamps (limit: 10).

- `units` (optional): Miles or kilometers.

TLE Data

- **URL**: `https://api.whereistheiss.at/v1/satellites/25544/tles`
- **Description**: Returns TLE data for the ISS.
- **Parameters**:
- `format` (optional): Response format (json or text). Default: json.

Coordinates Timezone Info

- **URL**: `https://api.whereistheiss.at/v1/coordinates/[lat,lon]`
- **Description**: Returns the timezone information, country code, and current time offset for a given set of coordinates.

Self evaluation

From the beginning our application was built for JavaFX and SceneBuilder. This means that .fxml files, their controllers and the application logic classes very naturally can be put into their own packages. Which are currently named as view, controller and model respectively.

Whether this is a true MVC-pattern is a little confusing, since in our application the application logic files, which correspond to model and .fxml files in view don't communicate with each other. Controllers handle all communication between them.

Doing it any other way would make it very difficult to prevent model and view classes from being tightly coupled, and therefore make our code much less portable. Right now, the architecture works well and there aren't any foreseeable problems on the horizon.

As of now all the problems have been due to JavaFX, and its poor/nonexistent support of for example Open Street Maps or Google Maps which forced us to make our world map implementation.

Despite that though, we have been able to follow our original design remarkably closely. Even if something was more difficult to implement than we originally thought, we haven't needed to give up anything yet.