Climate Monitoring

Technical Manual

Climate monitoring is an application designed to provide information to citizens regarding their area of interest about the weather conditions.
Operators have the task to update data by submitting information through parameters in the centers they are employed into.

Index

Download instructions

To clone the repository use the following git command:
`git clone https://github.com/adellafrattina/ClimateMonitoring.git`

Build instructions

Windows

- Make sure you have installed and available in PATH JDK 17 or higher
- Build client:
  - execute `scripts\maven\mvnw.cmd -am --projects climate-monitoring-client clean install` *or*
  - run `scripts\build\windows\build-client.bat`

Linux

- Install dependencies: openjdk17 or higher (package name could vary from system to system)
- Build client:
  - execute `scripts\maven\mvnw -am --projects climate-monitoring-client clean install` *or*
  - run `scripts\build\linux\build-client.bat`
- Build server:
  - execute `scripts\maven\mvnw -am --projects climate-monitoring-server clean install` *or*
  - run `scripts\build\linux\build-server.bat`

macOS

- Check dependencies from Linux section above and make sure you have them installed
- Build client:
  - execute `scripts\maven\mvnw -am --projects climate-monitoring-client clean install` *or*
  - run `scripts\build\macos\build-client.bat`
- Build server:
  - execute `scripts\maven\mvnw -am --projects climate-monitoring-server clean install` *or*
  - run `scripts\build\macos\build-server.bat`

System requirements

	Minimum	Recommended
OS (Windows)	32-bit Windows 10	64-bit windows 10 or higher
OS (Linux)	64-bit Linux distribution	64-bit Ubuntu 23.04 or higher, Arch Linux 2024.08.01 (X11)
OS (macOS)	64-bit macOS Monterey 12.01	64-bit macOS Monterey 12.01 or higher
OS (macOS arm64)	64-bit macOS Sonoma 14.6 (M1 CPU)	64-bit macOS Sonoma 14.6 (M1 CPU) or higher
Processor	Intel i5-7200U, AMD A8-6500	Intel i7-8750H, AMD Ryzen 5 3600
Memory	500 MB RAM	1.5 GB RAM
Graphics	Any integrated GPU that supports OpenGL	Discrete GPU that supports OpenGL
OpenGL	Version 3.0	Version 3.0 or higher
Storage	15 MB available space	30 MB available space or higher
Screen resolution	800 x 600	1920 x 1080 or higher

Note: *the specs listed above come from systems that have been used to run the application, if you don't see your system in the specs it means that it may or may not work*

Contribute to the development

1. Fork the repository
2. Clone the fork to your computer
   `git clone https://github.com/("your username")/ClimateMonitoring`
3. Create a new branch from the main one `git checkout -b ("your branch name")`
4. Write code and commit
5. Push commits to the remote repository `git push origin ("your branch name")`
6. Go on your fork and create a pull request
7. Wait for an admin to approve and merge your code
8. If your code was not approved, improve it and start from 4
9. If your code was approved, you can delete your branch and repeat the process from 3

Documentation

For more detailed and in-depth documentation regarding the database (conceptual and restructured E-R diagram, RASD, integrity constraints and translated logical schema) there are images and documents in the `/docs/database` directory.

In the parent directory `/docs` there is the application's RASD (Requirements Analysis and Specification Document) and a .mdj file (viewable with StarUML) that contains every UML diagram with more information compared to its .pdf counterpart.

Every UML diagram can be found in the `/docs/application` folder, as an easily accessible .pdf file.

Each Java class documentation will be in the `/docs/javadoc` folder, accessible with `index.html`

Design Patterns

- Singleton: used in the Application class and Handler class. The latter uses a standard approach of this pattern, where the former differs a little bit since it has a public constructor that needs to be called only once to create the actual static application instance. Calling the constructor a second time raises an error. This approach was chosen because the application can be either headless (CLI) or GUI.
- Proxy: used to abstract the client-server networking code from the rest of the application. It was chosen not to use Java's RMI approach since it's considered unreliable and hides away too many implementation details
- State: this approach is based on the original State pattern but with some changes to fit the project's needs. It is used for the client's views. This pattern provides an easy way to switch from one view to another with the call of just

one function. The usage of this pattern also allows to develop the views in parallel branches without any conflicts

- Adapter: this pattern is used to create a multithreaded version of the proxy module. The methods of this new class are non-blocking which is useful in GUI mode
- Façade: this pattern is used to abstract complex ImGui classes while making them into more user-friendly ones, which helped the development process of the views

Main directories

`/bin`: the directory where the resulting compiled client and server application are located

`/climate-monitoring-client`: the directory where the client's source code and the other resources, such as custom fonts and custom images are located

`/climate-monitoring-core`: the directory where the source code of the main components of the application, shared between client, server and widgets are located

`/climate-monitoring-server`: the directory where the server's source code is located

`/docs`: the directory where the documentation (manuals, UML diagrams, database diagrams, javadoc etc.) are located

`/scripts`: the directory where the building scripts are located for the three supported operating systems (Windows, Linux, macOS)

Data structures

The main data structure used to receive big lists of data are arrays. The main advantage is its constant time to access an element inside of it (always O(1)) and the fact that arrays are cache-friendly helps with performance, since they store their elements contiguously in memory.

Dependency tree