# PandemicTracker

A project by David, Mar, Daniel and Máté

# Problem:

No platform provides aggregate information on :

1. *Country preparedness*
2. *Travel bans*

∴ <u>High health cost</u> for the government during pandemic

∴ People can be more aware and <u>avoid infection</u>

# Solution:

Features & Functionalities:

1. **Data retrieval**

2. **Data storage**

3. **Report & Dashboard**
   a. Global and national

4. **Country comparisons**

# DEMO

# Process Model - Waterfall Model

- **Requirements well understood**: Our product is not being ordered by any external customers, but rather we are building it ourselves and offering it governments
  - We have complete control over requirements
- **Development is linear**: Our development followed a linear flow that begins with communication of requirements and ends with deployment.
  - There were no increments or expansions after the requirements were established.
- **Product is not growing or changing**: This product does not save any user information. Instead, its sole purpose is to scrape, organize, and display data.

# Division of Labor - Gantt Chart

# Implementation - Frontend

- Templates written in HTML
  - Javascript and CSS used in conjunction with Flask Jinja
- 5 main templates:
  - Home Page
  - Country List
  - Compare Countries
  - Single Country
  - Health Page
- Bootstrap used to polish interface
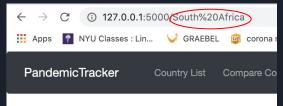
# Implementation - Frontend



**1.**



**2.**

# Implementation - Frontend



**3.**



**4.**

# Implementation - Back-end

- Framework: Flask
- Database: MySQL
- Flask + MySQL = Pymysql library
- Scraping: Python BeautifulSoup and Pandas

# Quality Assurance and Testing

- Each team member **wrote test cases** for the parts of the program that they wrote, to ensure that they could identify all possible defects.
- We tried our best to be cautious when fixing bugs: asking what other bugs a fix could introduce and doing **regression testing** after major fixes.
- Overall, we…
  - Made **13** test cases
  - Identified (and fixed!) **4** major bugs through those test cases
- The end result is a **robust backend** that can respond to and handle changing external circumstances, and a **responsive frontend** that seamlessly draws data from it.

# Future Maintenance

1. **Maintenance**
   a. PandemicTracker is **entirely dynamic**: as long as the data sources it relies on remain updated, it will present up-to-date data without any need for manual updating
   b. The data sources will need to be periodically checked for updatedness and accuracy
2. **Future features**
   a. Live deployment!
   b. Making scraping faster/more efficient
   c. Integration of more information