Main.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include "CSVReader.h"
#include "Tokeniser.h"
#include "CSVLine.h"
#include "CSVLineList.h"
#include "Candlestick.h"
#include "CandlestickPlotter.h"
#include <limits>
#include "Menu.h"

int main() {
    std::cout << "Starting..." << std::endl;

    // Read CSV file
    std::cout << "Reading CSV file" << std::endl;
    std::string csvFile = "opsd-weather_data-2020-09-16/weather_data.csv";
    CSVReader csvReader;
    std::vector<std::string> lines = csvReader.readCSV(csvFile);

    // Print number of lines
    std::cout << "Number of lines: " << lines.size() << std::endl;
    std::unordered_map<std::string, int> columnMap;
    std::vector<std::string> headerTokens = Tokeniser::tokenise(lines[0], ',');

    // Create columnMap
    int i = 0;
    for (std::string token : headerTokens)
    {
        columnMap[token] = i;
        i++;
    }

    // Create CSVLineList
    std::cout << "Creating CSVLineList" << std::endl;
    CSVLineList csvLineList;
    for (int i = 1; i < lines.size(); i++)
    {
        std::vector<std::string> tokens = Tokeniser::tokenise(lines[i], ',');
        CSVLine line(tokens, columnMap);
        csvLineList.addLine(line);
```

```
    }

    // Group lines by year
    csvLineList.groupByYear();

    // Initialise menu
    Menu menu(csvLineList);
    menu.run();
    std::cout << "Finishing..." << std::endl;
}
```

Tokeniser.h

```
#pragma once
#include <string>
#include <vector>

class Tokeniser {
    public:
        static std::vector<std::string> tokenise(std::string csvLine, char separator);
};
```

Tokeniser.cpp

```
#include <vector>
#include <string>
#include "Tokeniser.h"

std::vector<std::string> Tokeniser::tokenise(std::string csvLine, char separator)
{
    std::vector<std::string> tokens;
    std::string token;
    signed int start, end;
    start = csvLine.find_first_not_of(separator, 0);
    do
    {
        end = csvLine.find_first_of(separator, start);
        if (start == csvLine.length() || start == end)
            break;
        if (end >= 0)
            token = csvLine.substr(start, end - start);
        else
```

```
            token = csvLine.substr(start, csvLine.length() - start);
        tokens.push_back(token);
        start = end + 1;
    } while (end != std::string::npos);
    return tokens;
}
```

CSVReader.h

```
#pragma once
#include <string>
#include <vector>

class CSVReader {
    public:
        std::vector<std::string> readCSV(std::string csvFile);
};
```

CSVReader.cpp

```
#include "CSVReader.h"
#include <exception>
#include <string>
#include <vector>
#include <iostream>
#include <fstream>

std::vector<std::string> CSVReader::readCSV(std::string csvFilename)
{
    std::vector<std::string> lines;
    std::ifstream csvFile{csvFilename};
    std::string line;
    int lineNumber = 0;
    if (csvFile.is_open())
    {
        while (std::getline(csvFile, line))
        {
            try {
                lines.push_back(line);
            }catch(const std::exception& e)
            {
                std::cout << "Something went wrong on line " << lineNumber << std::endl;
            }
```

```
            lineNumber++;
        }
        csvFile.close();
    }
    else
    {
        std::cout << "Problem opening file " << csvFilename << std::endl;
    }
    return lines;
}
```

CSVLine.h

```
#pragma once
#include <string>
#include <vector>

class CSVLine {
    public:
        CSVLine(std::vector<std::string>& tokens, std::unordered_map<std::string, int>&
columnMap);
        std::string getDate();
        int getYear() const;
        double getTemperature(std::string country);
    private:
        std::vector<std::string> tokens;
        std::unordered_map<std::string, int> columnMap;
        int year;
        std::string date;
        double GB_temperature;
};
```

CSVLine.cpp

```
#include "CSVLine.h"
#include <vector>
#include <string>
#include <iostream>

CSVLine::CSVLine(std::vector<std::string>& _tokens, std::unordered_map<std::string,
int>& _columnMap) : tokens(_tokens), columnMap(_columnMap)
{
```

```
    date = tokens[0];
    std::string yearString = date.substr(0, 4);
    try{
        year = std::stoi(yearString);
    }
    catch(const std::exception& e)
    {
        std::cout << "Error converting year string to int" << std::endl;
    }
}

std::string CSVLine::getDate()
{
    return date;
}

int CSVLine::getYear() const
{
    return year;
}

double CSVLine::getTemperature(std::string countryTemperatureColumn)
{
    try{
        return std::stod(tokens[columnMap[countryTemperatureColumn]]);
    }
    catch(const std::exception& e)
    {
        std::cout << "Error converting values from csv line" << std::endl;
        std::cerr << "Error: " << e.what() << std::endl;
        throw std::invalid_argument("Invalid country temperature column");
    }
}
```

CSVLineList.h

```
#pragma once

#include "CSVLine.h"
#include <vector>
#include <map>

class CSVLineList {
```

```
    public:
        CSVLineList();
        std::vector<CSVLine> getLines();
        void addLine(CSVLine& line);
        std::map<int, std::vector<CSVLine> >& getLinesByYear();
        void groupByYear();
    private:
        std::vector<CSVLine> lines;
        std::map<int, std::vector<CSVLine> > linesByYear;
};
```

CSVLineList.cpp

```
#include "CSVLineList.h"
#include <map>
#include <iostream>

CSVLineList::CSVLineList()
{
}

std::vector<CSVLine> CSVLineList::getLines()
{
    return lines;
}

std::map<int, std::vector<CSVLine> >& CSVLineList::getLinesByYear()
{
    return linesByYear;
}

void CSVLineList::groupByYear()
{
    std::cout << "Grouping lines by year" << std::endl;
    for (CSVLine& line : lines) {
        int year = line.getYear();
        linesByYear[year].push_back(line);
    }
}

void CSVLineList::addLine(CSVLine& line)
{
    lines.push_back(line);
```

```
}
```

Candlestick.h

```
#pragma once
#include <string>
#include <vector>
#include "CSVLine.h"
#include "CountryFilter.h"

class Candlestick {
   public:
      Candlestick(std::vector<CSVLine>& lines, CountryFilter country, double open);
      std::string getDate() const;
      int getYear() const;
      double getClose() const;
      double getOpen() const;
      double getHigh() const;
      double getLow() const;
   private:
      std::string date;
      int year;
      CountryFilter country;
      std::vector<CSVLine>& lines;
      double open;
      double close;
      double calculateClose();
      double high;
      double calculateHigh();
      double low;
      double calculateLow();
};
```

Candlestick.cpp

```
#include "Candlestick.h"
#include "CSVLine.h"
#include <string>
#include <vector>
#include <limits>
```

```
Candlestick::Candlestick(std::vector<CSVLine>& _lines, CountryFilter _country, double
_open) : lines(_lines) {
  country = _country;
  date = lines[0].getDate();
  year = lines[0].getYear();
  close = calculateClose();
  high = calculateHigh();
  low = calculateLow();
  open = _open;
  if (open == std::numeric_limits<double>::lowest()) {
    open = close;
  }
}

std::string Candlestick::getDate() const { return date; }

int Candlestick::getYear() const { return year; }

double Candlestick::getOpen() const { return open; }

double Candlestick::getClose() const { return close; }

double Candlestick::getHigh() const { return high; }

double Candlestick::getLow() const { return low; }

double Candlestick::calculateClose() {
  double close = 0;
  for (CSVLine line : lines) {
    close += line.getTemperature(country.getAsColumnName());
  }
  return close / lines.size();
}

double Candlestick::calculateHigh() {
  double high = std::numeric_limits<double>::lowest();
  for (CSVLine line : lines) {
    double temp = line.getTemperature(country.getAsColumnName());
    if (temp > high) {
      high = temp;
    }
  }
  return high;
}
```

```
double Candlestick::calculateLow() {
  double low = std::numeric_limits<double>::max();
  for (CSVLine line : lines) {
    double temp = line.getTemperature(country.getAsColumnName());
    if (temp < low) {
      low = temp;
    }
  }
  return low;
}
```

CandlestickPlotter.h

```
#pragma once

#include <vector>
#include "Candlestick.h"

class CandlestickPlotter {
    public:
        CandlestickPlotter(std::vector<Candlestick>& _candlesticks);
        void plot();
    private:
        std::vector<Candlestick>& candlesticks;
};
```

CandlestickPlotter.cpp

```
#include <vector>
#include "Candlestick.h"
#include "CandlestickPlotter.h"
#include <iostream>
#include <iomanip>

// ANSI color codes
#define GREEN "\033[32m"
#define RED "\033[31m"
#define WHITE "\033[37m"
#define RESET "\033[0m"
```

```
CandlestickPlotter::CandlestickPlotter(std::vector<Candlestick>& _candlesticks):
candlesticks(_candlesticks)
{}

void CandlestickPlotter::plot()
{
    std::cout << "Plotting candlesticks" << std::endl;
    double minTemp = candlesticks[0].getLow();
    double maxTemp = candlesticks[0].getHigh();
    std::cout << "Candlesticks size: " << candlesticks.size() << std::endl;
    for (const Candlestick& candlestick : candlesticks)
    {
        minTemp = std::min(minTemp, candlestick.getLow());
        maxTemp = std::max(maxTemp, candlestick.getHigh());
    }
    std::cout << "Min temp: " << minTemp << " Max temp: " << maxTemp << std::endl;
    for (int temp = maxTemp; temp >= minTemp; temp--)
    {
        std::cout << std::setw(2) << std::setfill('0')<< temp << " ";
        for (const Candlestick& candlestick : candlesticks)
        {
            if (temp < candlestick.getLow() || temp > candlestick.getHigh())
            {
                std::cout << "     ";
                continue;
            }
            int boxTop = std::max(candlestick.getClose(), candlestick.getOpen());
            int boxBottom = std::min(candlestick.getClose(), candlestick.getOpen());
            if (temp <= boxTop && temp >= boxBottom)
            {
                if (candlestick.getClose() > candlestick.getOpen())
                {
                    std::cout << GREEN << " [ ] " << RESET;
                }
                else if (candlestick.getClose() < candlestick.getOpen())
                {
                    std::cout << RED << " [ ] " << RESET;
                }
                else
                {
                    std::cout << WHITE << " [=] " << RESET;
                }
            }
            else
```

```
            {
                if (candlestick.getClose() > candlestick.getOpen())
                {
                    std::cout << GREEN << "  |  " << RESET;
                }
                else if (candlestick.getClose() < candlestick.getOpen())
                {
                    std::cout << RED << "  |  " << RESET;
                }
                else
                {
                    std::cout << WHITE << "  |  " << RESET;
                }
            }
        }
        std::cout << std::endl;
    }
    // std::cout << std::endl;
    // Print x-axis
    std::cout << "    ";
    for (int i = 0; i < candlesticks.size(); i++)
    {
        std::cout << candlesticks[i].getYear() << " ";
    }
    std::cout << std::endl;
}
```

CountryFilter.h

```
#pragma once

#include <string>
#include <vector>

class CountryFilter
{
    public:
        CountryFilter();
        CountryFilter(const std::string& country);
        std::string getAsColumnName() const;
        std::string getCountry() const;
        static const std::vector<std::string> allowedCountries;
    private:
```

```
        std::string country;
        bool isAllowed(const std::string& country);
};
```

CountryFilter.cpp

```
#include "CountryFilter.h"
#include <string>
#include <vector>

CountryFilter::CountryFilter()
{
  country = "GB";
}
CountryFilter::CountryFilter(const std::string& _country)
{
  if (!isAllowed(_country))
  {
    throw std::invalid_argument("Invalid country");
  }
  country = _country;
}

const std::string countries[] = {
"AT", "BE", "BG", "CH", "CZ", "DE", "DK", "EE", "ES", "FI",
  "FR", "GB", "GR", "HR", "HU", "IE", "IT", "LT", "LU", "LV",
  "NL", "NO", "PL", "PT", "RO", "SE", "SI", "SK"
};

const std::vector<std::string> CountryFilter::allowedCountries(
    countries, countries + sizeof(countries)/sizeof(countries[0])
);

bool CountryFilter::isAllowed(const std::string& country) {
    return std::find(allowedCountries.begin(), allowedCountries.end(), country) !=
allowedCountries.end();
}

std::string CountryFilter::getCountry() const
{
  return country;
}
```

```
std::string CountryFilter::getAsColumnName() const
{
  return country + "_temperature";
}
```

DateRangeFilter.h

```
#pragma once

class DateRangeFilter {
   public:
      DateRangeFilter(int startYear, int endYear);
      int getStartYear() const;
      int getEndYear() const;
      bool isInRange(int year) const;
      bool isOneYearBefore(int year) const;
   private:
      int startYear;
      int endYear;
};
```

DateRangeFilter.cpp

```
#include "DateRangeFilter.h"
#include <stdexcept>

DateRangeFilter::DateRangeFilter(int _startYear, int _endYear): startYear(_startYear),
endYear(_endYear)
{
   if (startYear > endYear)
   {
      throw std::invalid_argument("Start year must be less than end year");
   }
   if (startYear < 1980 || endYear > 2025)
   {
      throw std::out_of_range("Date range out of range");
   }
}

int DateRangeFilter::getStartYear() const
{
   return startYear;
```

```
}

int DateRangeFilter::getEndYear() const
{
    return endYear;
}

bool DateRangeFilter::isInRange(int year) const
{
    return year >= startYear && year <= endYear;
}

bool DateRangeFilter::isOneYearBefore(int year) const
{
    return year == startYear - 1;
}
```

Menu.h

```
#pragma once

#include <string>
#include "CountryFilter.h"
#include "CSVLineList.h"
#include "DateRangeFilter.h"

enum MenuStatus {
    INITIALISING,
    EXIT,
    RUNNING
};

class Menu
{
    public:
        Menu(CSVLineList csvLineList);
        void run();
    private:
        CSVLineList csvLineList;
        MenuStatus status;
        CountryFilter countryFilter;
        DateRangeFilter dateRangeFilter;
        void printMenu();
```

```
        int getUserOption();
        void processUserOption(int userOption);
        void selectCountry();
        void selectDateRange();
        void exit();
        void plotChart();
        void predictTemperature();
};
```

Menu.cpp

```
#include "Menu.h"
#include <iostream>
#include "CountryFilter.h"
#include "Candlestick.h"
#include "CandlestickPlotter.h"
#include "TemperaturePredictor.h"

Menu::Menu(CSVLineList _csvLineList) :
csvLineList(_csvLineList),
dateRangeFilter(1980, 2025),
countryFilter("GB")
{
   status = MenuStatus::INITIALISING;
}

void Menu::run()
{
   status = MenuStatus::RUNNING;
   while (status == MenuStatus::RUNNING)
   {
      if (status != MenuStatus::RUNNING)
      {
         break;
      }
      printMenu();
      int userOption = getUserOption();
      processUserOption(userOption);
   }
}

void Menu::printMenu()
{
```

```
    std::cout << "=== MENU ===" << std::endl;
    std::cout << "Applied filters: " << std::endl;
    std::cout << "\t" << " selected Country: " << countryFilter.getCountry() << std::endl;
    std::cout << "\t" << " selected Date range: " << dateRangeFilter.getStartYear() << " - "
<< dateRangeFilter.getEndYear() << std::endl;
    std::cout << "Options: " << std::endl;
    std::cout << "1: Select country" << std::endl;
    std::cout << "2: Select date range" << std::endl;
    std::cout << "3: Plot chart" << std::endl;
    std::cout << "4: Predict temperature" << std::endl;
    std::cout << "5: Exit app" << std::endl;
    std::cout << "=== END MENU ===" << std::endl;
}

int Menu::getUserOption()
{
    int userOption = 0;

    std::string line;
    std::cout << "Type in 1-6" << std::endl;
    std::getline(std::cin, line);
    try {
        userOption = std::stoi(line);
    }catch(const std::exception& e) {
        std::cout << "Invalid choice. Choose 1-6" << std::endl;
    }
    std::cout << "You chose: " << userOption << std::endl;
    return userOption;
}

void Menu::processUserOption(int userOption)
{
    switch (userOption)
    {
        case 1:
            selectCountry();
            break;
        case 2:
            selectDateRange();
            break;
        case 3:
            plotChart();
            break;
        case 4:
```

```
            predictTemperature();
            break;
        case 5:
            exit();
            break;
        default:
            std::cout << "Invalid choice. Choose 1-1" << std::endl;
            break;
    }
}

void Menu::selectCountry()
{
    std::cout << "=== SELECT COUNTRY ===" << std::endl;
    std::cout << "Available countries: " << CountryFilter::allowedCountries[0];
    for (int i = 1; i < CountryFilter::allowedCountries.size(); i++) {
        std::cout << ", " << CountryFilter::allowedCountries[i];
    }
    std::cout << "." << std::endl;
    std::cout << "Enter country: ";
    std::string country;
    std::getline(std::cin, country);
    std::cout << "You chose: " << country << std::endl;
    try{
        countryFilter = CountryFilter(country);
        std::cout << countryFilter.getAsColumnName() << std::endl;
    } catch(const std::exception& e) {
        std::cout << "Invalid country" << std::endl;
    }
}

void Menu::exit()
{
    std::cout << "Exit app" << std::endl;
    status = MenuStatus::EXIT;
}

void Menu::plotChart()
{
    std::cout << "=== PLOT CHART ===" << std::endl;
    // Create candlesticks
    std::vector<Candlestick> candlesticks;
    double open = std::numeric_limits<double>::lowest();
    std::map<int, std::vector<CSVLine> >& linesByYear = csvLineList.getLinesByYear();
```

```
    for (auto it = linesByYear.begin(); it != linesByYear.end(); it++)
    {
        if (dateRangeFilter.isOneYearBefore(it->first))
        {
            Candlestick candlestick(it->second, countryFilter, open);
            open = candlestick.getClose();
            continue;
        }
        if (!dateRangeFilter.isInRange(it->first))
        {
            continue;
        }
        Candlestick candlestick(it->second, countryFilter, open);
        candlesticks.push_back(candlestick);
        open = candlestick.getClose();
        std::cout << "Candlestick: " << candlestick.getDate() << " " <<
candlestick.getOpen() << " " << candlestick.getClose() << " " << candlestick.getHigh() <<
" " << candlestick.getLow() << std::endl;
    }
    // Plot candlesticks
    CandlestickPlotter candlestickPlotter(candlesticks);
    candlestickPlotter.plot();
}

void Menu::selectDateRange()
{
    std::cout << "=== SELECT DATE RANGE ===" << std::endl;
    std::cout << "Enter start year: ";
    std::string startYear;
    std::getline(std::cin, startYear);
    std::cout << "Enter end year: ";
    std::string endYear;
    std::getline(std::cin, endYear);
    try{
        dateRangeFilter = DateRangeFilter(std::stoi(startYear), std::stoi(endYear));
    } catch(const std::invalid_argument& e) {
        std::cout << "Invalid date range: " << e.what() << std::endl;
    } catch(const std::out_of_range& e) {
        std::cout << "Date range out of range: " << e.what() << std::endl;
    }
}

void Menu::predictTemperature()
{
```

```
    std::cout << "=== TEMPERATURE PREDICTION ===" << std::endl;
    std::cout << "Selected Country: " << countryFilter.getCountry() << std::endl;
    std::cout << "Enter number of years to predict (1-5): ";
    std::string input;
    std::getline(std::cin, input);
    int yearsToPredict = 0;
    try {
        yearsToPredict = std::stoi(input);
    } catch(const std::exception& e) {
        std::cout << "Invalid input. Please enter a number between 1-5." << std::endl;
        return;
    }
    if (yearsToPredict < 1 || yearsToPredict > 5) {
        std::cout << "Invalid range. Please enter a number between 1-5." << std::endl;
        return;
    }
    std::cout << "Analyzing historical data..." << std::endl;
    try {
        TemperaturePredictor predictor(csvLineList, countryFilter);
        predictor.displayPredictions(yearsToPredict);
    } catch(const std::exception& e) {
        std::cout << "Error during prediction: " << e.what() << std::endl;
    }
}
```

TemperaturePredictor.h

```
#pragma once

#include <vector>
#include <utility>
#include <string>
#include "CSVLineList.h"
#include "CountryFilter.h"
/**
    This class is used to predict the temperature of a country for a given number of years.
    It uses the simple method of calculating the average yearly temperature change and
then predicting the temperature for the given number of years.
    It does not use any machine learning or other complex methods.
 */
class TemperaturePredictor {
public:
    TemperaturePredictor(CSVLineList& csvLineList, CountryFilter& country);
```

```
    std::vector<std::pair<int, double> > predict(int yearsToPredict);
    void displayPredictions(int yearsToPredict);

private:
    std::vector<std::pair<int, double> > historicalData; // year, avg_temp
    double yearlyChange;    // How much temperature changes per year
    double lastYearTemp;    // Temperature of the most recent year
    CountryFilter country;

    void extractHistoricalData(CSVLineList& csvLineList);
    void calculateYearlyChange();
};
```

TemperaturePredictor.cpp

```
#include "TemperaturePredictor.h"
#include "Candlestick.h"
#include <iostream>
#include <iomanip>
#include <cmath>
#include <limits>

TemperaturePredictor::TemperaturePredictor(CSVLineList& csvLineList, CountryFilter&
_country)
    : country(_country), yearlyChange(0.0), lastYearTemp(0.0) {
    extractHistoricalData(csvLineList);
    calculateYearlyChange();
}

void TemperaturePredictor::extractHistoricalData(CSVLineList& csvLineList) {
    std::map<int, std::vector<CSVLine> >& linesByYear = csvLineList.getLinesByYear();
    for (auto it = linesByYear.begin(); it != linesByYear.end(); ++it) {
        int year = it->first;
        std::vector<CSVLine>& yearLines = it->second;
        if (!yearLines.empty()) {
            double dummyOpen = std::numeric_limits<double>::lowest();
            Candlestick candlestick(yearLines, country, dummyOpen);
            double avgTemp = candlestick.getClose();
            historicalData.push_back(std::make_pair(year, avgTemp));
        }
    }
}
```

```
void TemperaturePredictor::calculateYearlyChange() {
    double firstYearTemp = historicalData.front().second;
    int firstYear = historicalData.front().first;
    lastYearTemp = historicalData.back().second;
    int lastYear = historicalData.back().first;
    int totalYears = lastYear - firstYear;
    yearlyChange = (lastYearTemp - firstYearTemp) / totalYears;
}

std::vector<std::pair<int, double> > TemperaturePredictor::predict(int yearsToPredict) {
    std::vector<std::pair<int, double> > predictions;
    int latestYear = historicalData.back().first;
    for (int i = 1; i <= yearsToPredict; ++i) {
        int futureYear = latestYear + i;
        double predictedTemp = lastYearTemp + (yearlyChange * i);
        predictions.push_back(std::make_pair(futureYear, predictedTemp));
    }
    return predictions;
}

void TemperaturePredictor::displayPredictions(int yearsToPredict) {
    std::vector<std::pair<int, double> > predictions = predict(yearsToPredict);
    std::cout << "Country: " << country.getCountry() << std::endl;
    std::cout << "Last recorded temperature: " << lastYearTemp << "°C" << std::endl;
    std::cout << "Yearly temperature change: " << yearlyChange << "°C per year" <<
std::endl;
    std::cout << "Predicted temperature for " << yearsToPredict << " years." << std::endl;
    std::cout << "Predicted Average Temperatures:" << std::endl;
    std::cout << std::fixed << std::setprecision(2);
    for (const auto& prediction : predictions) {
        std::cout << prediction.first << ": " << prediction.second << "°C" << std::endl;
    }
}
```