

Population Report System

Checklist 3



|  |  |
| --- | --- |
| Name | ID |
| Syed Abdul Samad Zaidi | 40692317 |
| Muhammad Ali Khalid | 40692222 |
| Muhammad Taha Khalid | 40692271 |
| Hamza Shahid | 40692218 |

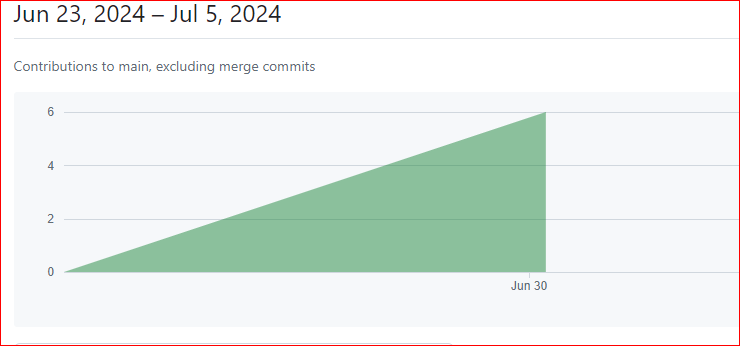
**https://github.com/Hamzashahid08/population\_report\_generator**

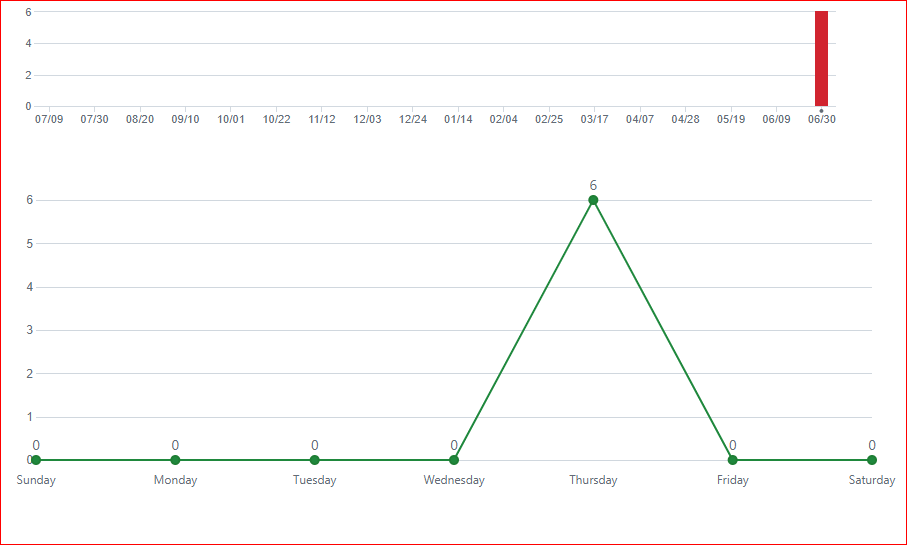
* **Metrics from GitHub**

To evaluate individual contributions and overall project activity, we reviewd following github metrices

 **Commit Frequency and Contributions**:

* The Contributors section in GitHub Insights was analyzed to understand the commit frequency of each team member.
* Detailed commit history was reviewed to assess the changes made by each contributor.



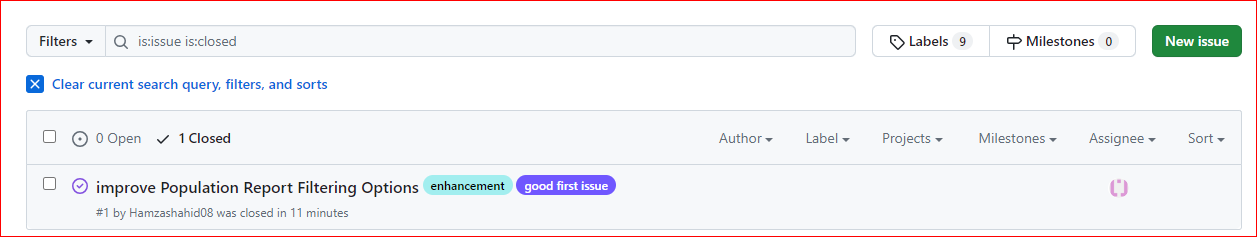


 **Pull Requests**:

* The Pull Requests tab was examined to see the number of pull requests created, discussions held, and reviews conducted.
* This helped in understanding the collaborative efforts and code review process.

 **Issues and Resolution**:

* The Issues tab provided insights into the number of issues reported and resolved.
* Details about who reported, who resolved, and the status of each issue were documented.



* **Code Quality Including Comments**

1. **Code Structure and Readability:**

* The project demonstrates a well-organized code structure, with clear separation of concerns. Each function handles specific tasks related to database queries or report generation.
  + Meaningful variable names contribute significantly to code readability. For instance, using “N” to represent the limit value is concise and intuitive.

1. **Code Comments and Documentation:**

* Inline comments play a crucial role in explaining complex logic and SQL queries. They serve as a guide for future maintainers, helping them understand the code’s intricacies.
* To enhance documentation further, consider adding high-level comments at the beginning of each method. These summaries can provide an overview of the method’s purpose, input parameters, and expected output

**Chunck of CODE to show the comments and structure:**

// Method to display countries by population in descending order

static void DisplayCountriesByPopulation()

{

string sqlQuery = "SELECT Name, Population FROM country ORDER BY Population DESC";

ExecuteAndDisplayQuery(sqlQuery);

}

// Method to display cities by population in descending order

static void DisplayCitiesByPopulation()

{

string sqlQuery = "SELECT Name, Population FROM city ORDER BY Population DESC";

ExecuteAndDisplayQuery(sqlQuery);

}

// Method to display capital cities by population in descending order

static void DisplayCapitalCitiesByPopulation()

{

string sqlQuery = "SELECT city.Name, city.Population FROM city JOIN country ON city.ID = country.Capital ORDER BY city.Population DESC";

ExecuteAndDisplayQuery(sqlQuery);

}

// Method to display top N populated countries

static void DisplayTopNPopulatedCountries(int n)

{

string sqlQuery = $"SELECT Name, Population FROM country ORDER BY Population DESC LIMIT {n}";

ExecuteAndDisplayQuery(sqlQuery);

}

// Method to display top N populated cities

static void DisplayTopNPopulatedCities(int n)

{

string sqlQuery = $"SELECT Name, Population FROM city ORDER BY Population DESC LIMIT {n}";

ExecuteAndDisplayQuery(sqlQuery);

}

// Method to display top N populated capital cities

static void DisplayTopNPopulatedCapitalCities(int n)

{

string sqlQuery = $"SELECT city.Name, city.Population FROM city JOIN country ON city.ID = country.Capital ORDER BY city.Population DESC LIMIT {n}";

ExecuteAndDisplayQuery(sqlQuery);

}

### **Correct Usage of Branches**

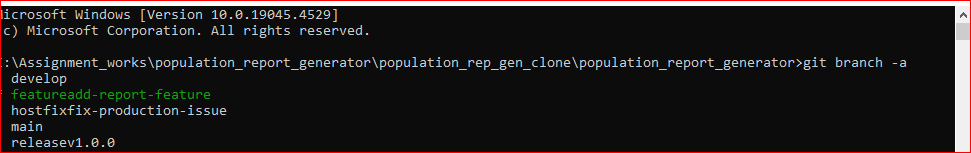
To ensure effective usage of Git branches according to the established GitFlow strategy, the following steps were performed:

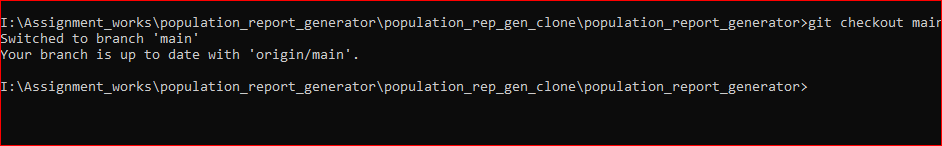
1. **Branch Structure Review**:
   * Confirmed existence of key branches: main, develop, feature, release, and hotfix.
2. **Branch Histories Check**:
   * Verified correct usage through commit histories:
     + main: Stable releases.
     + develop: Latest development changes.
     + feature: New features.
     + release: Preparing releases.
     + hotfix: Urgent bug fixes.
3. **Merge Practices Verification**:

* Ensured merges follow GitFlow strategy.
* Checked merge commits for proper practices.

1. **Pull Requests Evaluation**:

* Reviewed pull requests on GitHub:
  + - Used for merging changes.
    - Reviewed and approved by team members.
    - Included descriptive titles, summaries, and issue references.





* **Kanban/Project Board Usage**

1. **Integration**:

* Successfully integrated the Zube.io Kanban board with our GitHub repository.

1. **Task Import**:

* Imported all existing GitHub issues into Zube.io to centralize task tracking.

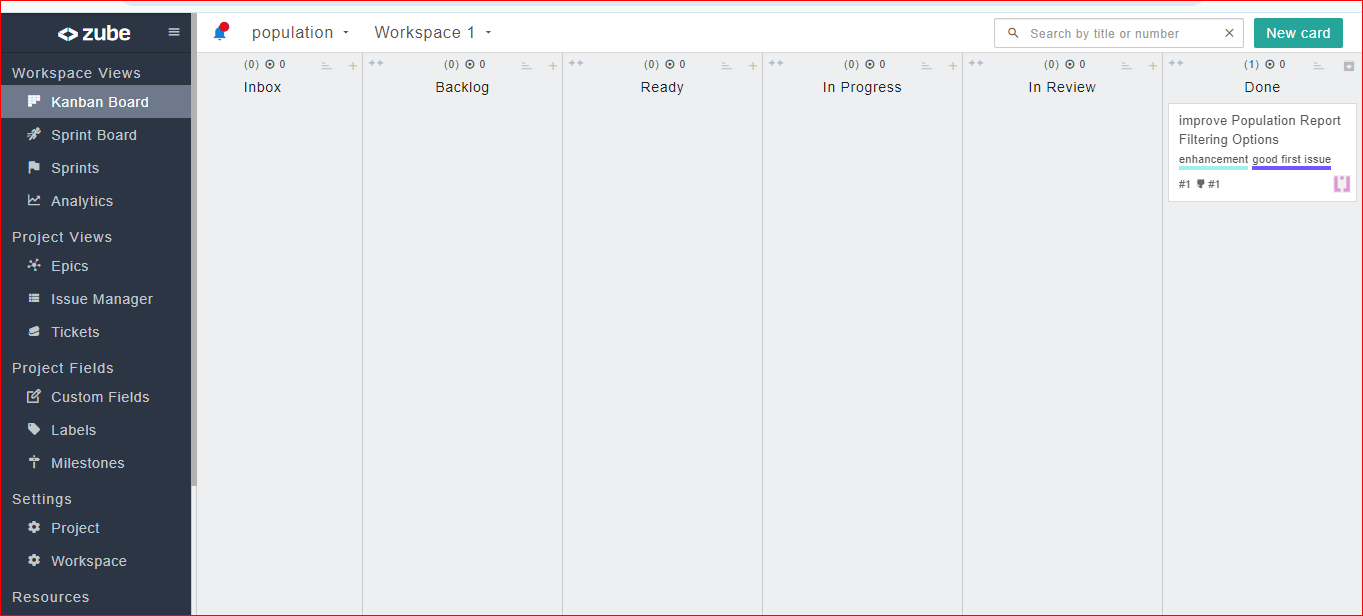
1. **Board Columns**:

Defined the following columns:

* + - **Backlog**: Tasks pending prioritization.
    - **To Do**: Tasks ready for development.
    - **In Progress**: Tasks currently being worked on.
    - **Review**: Tasks pending review and approval.
    - **Done**: Completed tasks.

1. **Task Management**:

* Moved tasks across columns based on their status.
* Assigned tasks to team members to ensure accountability and efficient task management.



* **Docker Integration**

.

1. **Dockerfile Creation**:

* A Dockerfile was added to the project directory.
* The Dockerfile included specifications for the base image, dependencies, and other configurations required for the C# application.

1. **Building and Running Docker Container**:



1. **Testing**:

* Verified the Docker container was running correctly and the application was accessible.

**Dockerfile:**

# Use the official .NET SDK image as the base image

FROM mcr.microsoft.com/dotnet/sdk:latest AS build

# Set the working directory inside the container

WORKDIR /app

# Copy the necessary project files into the container

COPY . /app

# Restore the dependencies

RUN dotnet restore

# Build the project

RUN dotnet build -c Release -o /app/build

# Publish the project

RUN dotnet publish -c Release -o /app/publish

# Use the official .NET runtime image as the base image for runtime

FROM mcr.microsoft.com/dotnet/aspnet:latest AS runtime

# Set the working directory inside the container

WORKDIR /app

# Copy the published output from the build stage to the runtime stage

COPY --from=build /app/publish .

# Install MySQL client library

RUN apt-get update && apt-get install -y \

default-libmysqlclient-dev \

&& rm -rf /var/lib/apt/lists/\*

# Expose the port on which your application will run (if applicable)

# EXPOSE 8080

# Set the default command to run the application

CMD ["dotnet", "YourApp.dll"]

* **Project Requirements Met**

**Feature Implementation**

1. **View Countries and Cities by Population:**

* Users can see a list of countries and cities sorted by population.
* Acceptance Criteria: The system displays this information clearly.

1. **View Top N Populated Cities and Capital Cities:**

* Users specify N and get the most populated cities and capital cities.
* Acceptance Criteria: The system allows N input and shows the relevant cities.

1. **Generate Language Speakers Statistics:**

* Users can explore language diversity stats (number of speakers, percentages).
* Acceptance Criteria: The system calculates and displays these statistics.

1. **Generate Population Breakdown Report by Continent:**

* Users get a breakdown of population by continent .
* Acceptance Criteria: The system generates and presents this report.

**Documentation and Code Quality:**

* The codebase is well-documented with meaningful comments.
* Major functions and classes have comprehensive explanations.
* Code quality adheres to readability and maintainability standards.