. GITHUB ANALYTICS DASHBOARD

PROJECT DELIVERABLE-1 & 2

SI.No	Name	A-number
1	Akshitha Bedre Shivakumar	A20544641
2	Arpitha Ramakrishnaiah	A20523821
3	Lokareddy Prishitha Reddy	A20555934

COURSE: SOFTWARE PROJECT MANAGEMENT (CS587)

DATE: 12TH NOVEMBER 2023

CONTENTS:

1) PROJECT CHARTER

1.1 OBJECTIVE

The objective of a GitHub analytics dashboard is to offer a centralized and visually accessible summary of important metrics and data pertinent to a software development project or repository on GitHub. This dashboard is designed to assist a range of users, including developers, project managers, and stakeholders, in gaining insights into how the project is performing, understanding collaboration dynamics, and assessing its overall status. By presenting data like the project's commit history, issue tracking, pull request activity, and code review statistics, the dashboard empowers users to make well-informed decisions, monitor progress, pinpoint obstacles, and enhance the efficiency of the development process. It functions as a valuable tool for improving productivity, transparency, and project management within the GitHub-based software development context.

1.2 SCOPE

The scope of the GitHub analytics dashboard serves as a centralized hub for monitoring and assessing the various aspects of a repository or organization's software development and collaboration efforts. It offers insights into key metrics, such as code contributions, issue tracking, pull requests, and release management. Users can visualize trends, track progress, and identify areas that may need attention, ultimately helping them make data-informed decisions to improve the development process, ensure code quality, and enhance collaboration among contributors. This dashboard provides a bird's-eye view of the health and productivity of a project, making it an essential tool for project managers, developers, and stakeholders to keep projects on track and achieve their goals efficiently.

1.3 STAKEHOLDERS

- Developers
- Project Managers
- Team Leads
- Executives

2. PROJECT PLAN

2.1 PROJECT TIMELINE

Week 1: Project Initiation
Assign roles to project teams (Project Manager, Developers, QA, etc.). Create a project charter that outlines the high-level requirements, stakeholders, project objectives, and scope.

Week 2: Compiling the Needs
To obtain specific requirements, survey, and interview stakeholders.
Include both functional and non-functional needs in a requirements document.

Week 3: Design Phase
Design the dashboard architecture for the GitHub Analytics Dashboard based on the requirements that have been gathered. Describe the user interfaces, data flow, and technological stack.

Week 4: Project Planning
Make a thorough project plan that addresses the resources, timetable, scope, and risk assessment. Create the project documentation and describe the development and testing strategy.

Week 5: Initial Development
Start building the GitHub dashboard for visualizations in accordance with the project schedule. Put basic database design and system structures into practice.

Week 6: Feature Implementation and Testing
Apply essential features while adhering to best practices and code standards. Once a feature is finished, test it unit-wise.

Week 7: Monitoring, Control, and Reflection
Use project management software to monitor and control progress. Take appropriate security precautions and address any new scope adjustments or cybersecurity threats.

Week 8: Project Finalization
Finish the testing procedures and outstanding development tasks.
Assign responsibilities to team members to overcome any resource limitations. Present the finished product to the stakeholders, get their input, and create a thorough project report that

details the tactics, obstacles, and results

2.1 BUDGET

- Development Team Expenses:Compensation for developers, designers, project managers, and quality control experts.
- Infrastructure and Hosting:Expenditures related to server hosting or cloud services like AWS, Azure, or GitHub Pages.
- Database Costs:Payments and Expenses for database software licenses or cloud-based database services,maintainence and administration.
- APIs and Integrations:Charges for third-party APIs and services utilized for data collection and costs for developing and integrating custom APIs if necessary.
- Testing and Quality Assurance:Expenses for quality assurance testing, which includes both manual and automated testing.
- Project Management and Miscellaneous Expenses:Miscellaneous outlays, such as design assets, documentation, or training.
- Support and Maintenance:Ongoing financial commitments for support and maintenance beyond the initial development phase.

2.3 RISKS AND ASSUMPTIONS:

Risks:

- Storing sensitive data on the dashboard can create security vulnerabilities, potentially exposing data to breaches.
- Inaccurate or incomplete data may lead to incorrect insights.
- A complex dashboard can overwhelm users, hindering insight extraction.
- The dashboard must scale with project growth to avoid performance issues.
- Regular updates are critical; neglect can harm the user experience.
- Relying on external services can affect dashboard functionality.
- Non-compliance may lead to legal issues.
- Underestimating costs can strain resources.
- Unclear objectives may hinder value delivery.
- Users may resist workflow changes.
- Poor design affects user satisfaction and usability.

Assumptions:

- Anticipating that users will actively interact with and make use of the dashboard to gain insights.
- Ensuring the availability of necessary resources, such as a development team and

tools.

- Taking for granted that the data collected from GitHub is both accurate and up-to-date.
- Assuming that adequate security measures are in position to safeguard sensitive data.
- Expecting that users have access to training or documentation to proficiently utilize the dashboard.
- Presuming that stakeholders share common goals and expectations for the dashboard

3. REQUIREMENTS

3.1 USER REQUIREMENTS

- Users require a concise summary of project information, encompassing code contributions, issue tracking, and pull requests.
- Users should have the flexibility to personalize the dashboard to their specific preferences, including adding or removing widgets as needed.
- The dashboard should furnish data in real-time or near-real-time, enabling users to monitor project advancements.
- Users seek charts, graphs, and visual aids to swiftly grasp data trends.
- The dashboard should be usable and effective on mobile devices, allowing for on-the-fly monitoring.
- It is essential that the dashboard is user-friendly and easy to navigate.
- Users should have the capability to configure alerts for pivotal events and changes.

3.2 FUNCTIONAL REQUIREMENTS:

- Secure Access: The dashboard should require secure user authentication to safeguard project information.
- Integrated Data: Seamlessly integrate data from GitHub repositories and other relevant sources into the dashboard.
- Visual Data Representation: Employ charts, graphs, and tables to present data clearly and concisely.
- Customizable Interface: Empower users to personalize their dashboards by adding, removing, or rearranging widgets.

- Real-Time Data Sync: Ensure data updates continuously or at frequent intervals to reflect current project status.
- Issue Management: Provide tools to monitor, track, and manage GitHub issues and pull requests effectively.
- Data Protection: Implement robust security measures to protect sensitive project data from unauthorized access.
- User Access Control: Define user roles with varying levels of access and permissions to ensure appropriate data access.
- Proactive Alerts: Implement a notification system to alert users about critical events and changes within the project.
- Data Export: Enable users to export data for further analysis or reporting to external platforms.

3.3 NON-FUNCTIONAL REQUIREMENTS:

- Efficiency:
  The dashboard should be able to handle a high volume of users and simultaneous requests without experiencing slowdowns or glitches.
  Data should be retrieved and loaded quickly, even for complex queries or visualizations.
  The dashboard should be responsive and interactive, allowing users to easily navigate and interact with data without delays or lag.
- Dependability:
  The dashboard should be consistently available and reliable, with minimal downtime or service interruptions.
  Data should be accurate and up-to-date at all times, with minimal errors or data loss.
  The dashboard should be able to quickly recover from unexpected errors or technical issues.
- Expandability:
  The dashboard should be able to adapt and grow to accommodate increasing amounts of data and user traffic.
  The underlying architecture should be designed to support future growth and expansion.
  The dashboard should be able to handle increasing workloads without experiencing performance bottlenecks.
- Ease of Use:
  The dashboard should be intuitive and easy to use, even for users with limited

technical knowledge or experience.
The interface should be user-friendly and organized, with clear labels and consistent design patterns.
The dashboard should provide comprehensive documentation and support resources for users.

- Data Protection:
  The dashboard should implement robust security measures to protect sensitive data from unauthorized access, theft, or misuse.
  User authentication and authorization mechanisms should be strong and reliable.
  Data should be encrypted both at rest and in transit to prevent unauthorized access or interception.
  The dashboard should be regularly scanned for vulnerabilities and patched as needed to maintain a high level of security.

4. DESIGN DOCUMENTS

4.1 SYSTEM ARCHITECTURE

- Frontend: The user interface where data is displayed and interacted with. It's usually built using HTML and CSS
- Backend: This handles data processing, storage, and business logic. The dashboard id bult on XAMML,SQL
- API: Create APIs for data retrieval and update. RESTful or GraphQL APIs are common choices.
- Visualization: Use libraries like D3.js or charting tools (e.g., Chart.js) for creating interactive data visualizations

4.2 DATABASE DESIGN

This database design allows to store from repositories, track retention-related data, record important events, and gather survey data for the dashboard. The database design in this case includes the name ,URL, stars ,forks ,watchers values.

4.3 USER INTERFACE DESIGN

The user interface design includes Login Page, Dashboard home, GitHub Repositories which consists of repos, checking of important alerts ,visualizations for graphs and Logging out .
Some of the additional design considerations are,

- Implementing role-based access control for different user types.
- Using intuitive icons and buttons for actions (e.g., search, edit, delete).
- Applying data visualization techniques for clear and informative charts/graphs.

5. DEVELOPMENT DOCUMENTATION

5.1 CODING GUIDELINES

- Use clear and consistent variable, function, and class naming conventions to make your code more readable.
- Organize your code into modular components, making it easier to manage and maintain.
- Implement robust error handling and display informative error messages to users when something goes wrong.
- Ensure that your dashboard is responsive and works well on different devices and screen sizes.
- Optimize database queries, use indexes, and avoid redundant data storage to improve performance.
- Use version control (e.g., Git) to track changes and collaborate with other developers.
- Provide meaningful feedback to users to enhance user experience.

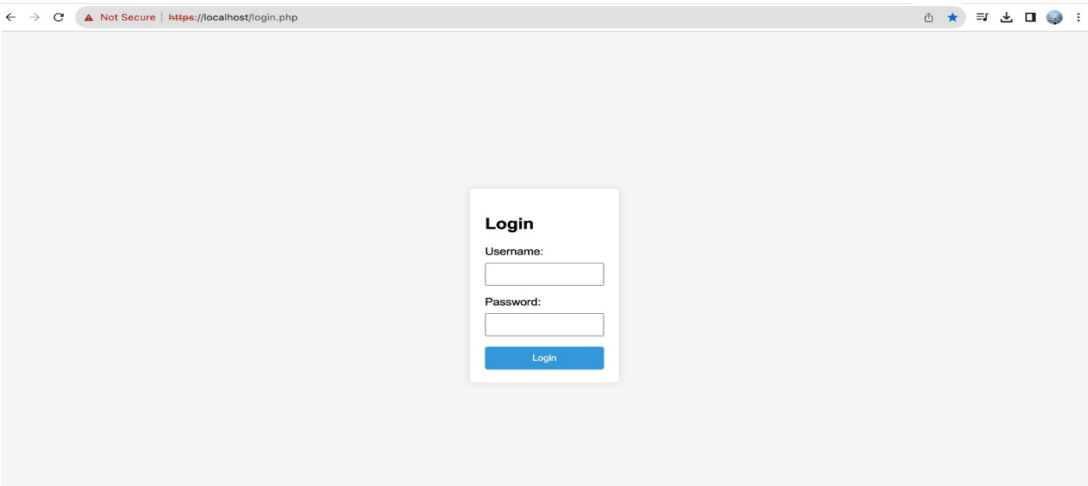
5.2 CODE REVIEWS AND VERSION CONTROL

- Code Reviews enhances code quality and collaboration.
- Version Control ensures the safe and organized management of the codebase, offering history tracking, collaboration support, and effective conflict resolution.
- Version control facilitates collaboration by allowing multiple developers to work on the same codebase simultaneously and merge their changes without conflicts.
- Reviewing code for security vulnerabilities, performance optimizations, and adherence to coding standards.
- Address issues and provide constructive feedback to maintain code quality.
- Uses code review tools or platforms (e.g., GitHub) to streamline the process.
- Prioritize code reviews for critical or sensitive parts of the system, such as authentication and data handling.
- Utilizes a version control system like Git to track changes and collaborate effectively.
- Maintains a clean and well-documented commit history to aid in tracking changes and debugging.
- Tag important milestones and releases for clarity and easy rollbacks.

6. DEPLOYMENT AND IMPLEMENTATION

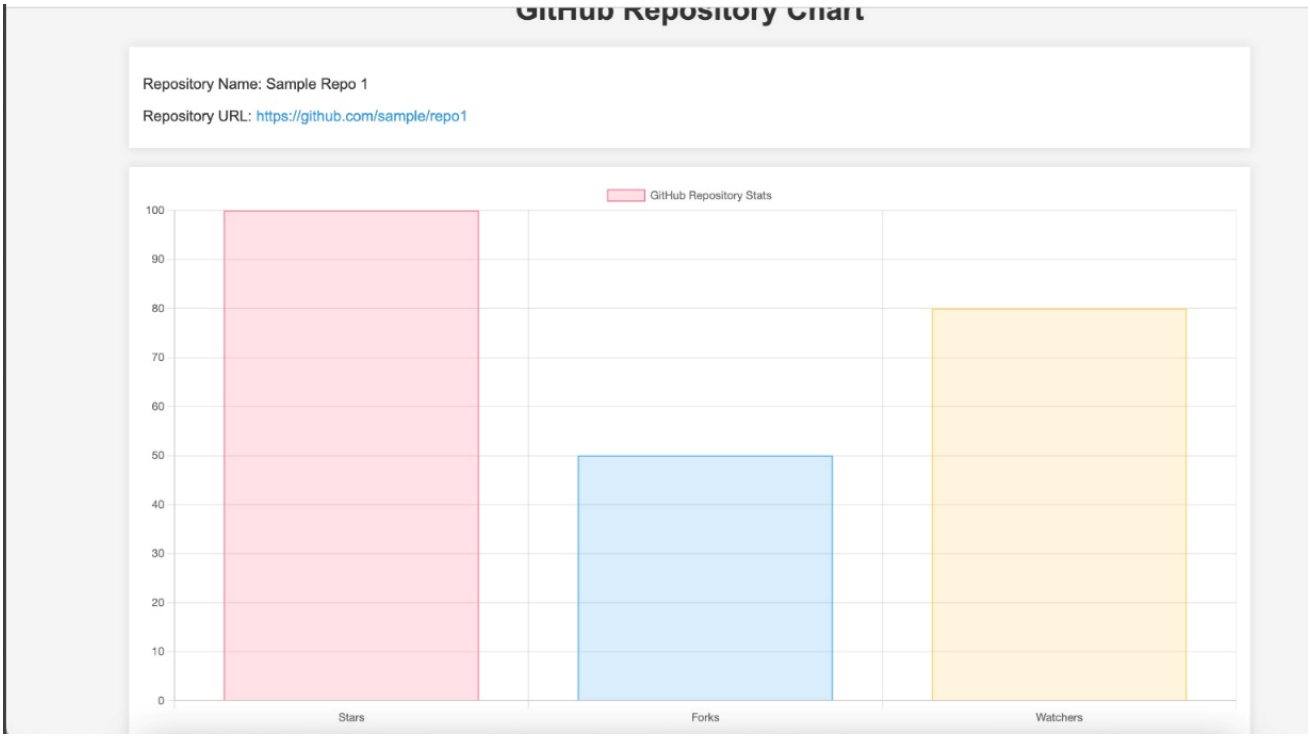
The deployment and Implementation have the following specific steps:

- Setting a web server by choosing the web host provider
- Obtaining a domain name for the dashboard to encrypt data transmission.
- Organizing the project files into a clear directory structure for maintaining.
- The frond end technologies HTML and CSS are used for writing the code and styling of the application.
- XAMMP web application environment is used for writing queries for the database in MYSQL for the Dashboard.
- Connected to the GitHub API to retrieve data about GitHub repositories, issues, and pull requests.
- The web development
- We implemented backend and frontend for data security and responsive design.
- While executing we ‘ll get the URL which directs us to the GitHub repositories we have created and it shows the visualization of the charts which are shown below.
- The GitHub shows that the repositories are being hosted on local server
- The chart suggests that the repository is active and that developers are regularly opening issues and pull requests to contribute to the project.
- Document the code and system configurations to aid future maintenance and troubleshooting is done.
- Regularly update and maintain the system if any new features and improve them.



GitHub Repositories

ID	Name	URL	View Chart	Stars	Forks	Watchers	Created At
1	Sample Repo 1	https://github.com/sample/repo1	View Chart	100	50	80	2023-11-11 20:40:17
2	Sample Repo 2	https://github.com/sample/repo2	View Chart	150	70	120	2023-11-11 20:40:17
3	Sample Repo 3	https://github.com/sample/repo3	View Chart	60	10	20	2023-11-11 21:50:27
4	Sample Repo 4	https://github.com/sample/repo4	View Chart	100	30	40	2023-11-11 21:50:27



7. TESTING AND QUALITY ASSURANCE

7.1 TESTING

- Unit Testing: Evaluated individual components and modules within the dashboard application to confirm their correct functionality.
- Integration Testing: Examined the collaboration between various components and modules in the dashboard application to guarantee smooth interactions.
- User Acceptance Testing (UAT): Assessed the dashboard application with actual users to verify that it fulfills their needs and is user-friendly.

7.2 QUALITY ASSURANCE

- Code Evaluation: Examined the dashboard code for potential errors, security risks, and design issues.
- Performance Assessment: Evaluated the dashboard application's performance by subjecting it to heavy loads to confirm its ability to manage numerous concurrent users and requests without performance degradation.
- Security Evaluation: Examined the dashboard application for security vulnerabilities and take necessary measures to address them.

8.RISK REGISTER

A risk register is basically a document utilized to recognize, evaluate, and oversee risks. It serves as an instrument enabling organizations to actively confront possible issues and guarantee their readiness to handle them proficiently.

While working on our GitHub Analytics Dashboard we experienced risk resources such as

- Failed to understand the requirements accurately
- Time-constraint and timeline
- Not having proper communication
- Learning aspect developed between our team
- Data accuracy and data volume were the main concerns
- Resource availability have been risk factor
- Obtaining valuable user feedback for enhancements can be challenging if users are
- not actively involved or if feedback mechanisms are not firmly established.

9.MAINTAINENCE AND SUPPORT

- Regular Updates: Continuously updating and patch the system to fix bugs and security vulnerabilities.
- Browser Compatibility: Ensuring that the system works smoothly on different web browsers and devices.
- Performance Optimization: Periodically optimizing code and assets for speed and efficiency.
- Data Backup: Regularly back up user data and system configurations to prevent data loss.
- User Support: Provide responsive user support to address user inquiries and issues.\
- Compliance Checks: Regularly review and update compliance with legal and regulatory requirements.
- Testing and QA: Conduct periodic testing and quality assurance to maintain system reliability.

10.FINAL PROJECT REPORT

- Our team updated the project from time to time.
- Any risks or vulnerabilities related to the tool will be documented and fixed through regular updates to the dashboard
- Held regular meetings weekly and updated what our roles and what part each of one was
- Responsible in doing
- Leaned about GitHub Dashboard more in detail while developing it .
- Explored new technologies when we were researching about GitHub Analytics Dashboard
- The dashboard we developed provided valuable insights such as Project progress ,project stats ,contributing collaboration.
- The dashboard had been tested and concluded as to be secure ,reliable and accurate.
- Any risks or vulnerabilities related to the tool will be documented and fixed through regular updates

We are attaching the local host id : https://localhost/phpmyadmin/
Login page : https://localhost/login.php
Id: admin
Password: password123