Software Design Specifications

for

Project XLR8 – Version 1.0.1

Prepared by:

Team 35 -

- SE21UARI049 – Hemanth Kakkireni
- SE21UARI057 – Jatin Mamidi
- SE21UARI065 – Sahan Katpally
- SE21UARI069 – Krishna Abhirama Somayajula
- SE21UARI085 – Bhargav Mulakala
- SE21UARI087 – Shashank Mutyala
- SE21UARI089 – Srinivas Nalla
- SE21UARI112 – Ramki Jetti
- SE21UARI130 – Sai Varun Padmanabham

Document Information

Title: Project XLR8	
Project Manager: **Shashank Mutyala** **Srinivas Nalla** **Ramki Jetti**	**Document Version No: 1.0.1** **Document Version Date: 14-04-2024**
Prepared By: **Jatin Mamidi,** **Kakkireni Hemanth,** **Krishna Abhirama,** **Sai Varun Padmanabham**	**Preparation Date: 10-04-2024**

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename
1.0	10-04-2024	Jatin Mamidi, Kakkireni Hemanth, Krishna Abhirama, Sai Varun Padmanabham	First update of the document, included with all details of the current base website	Project Software Design Specification - Team 35
1.0.1	14-04-2024	Kakkireni Hemanth, Sahan Katpally, Bhargav Mulakala	Added information regarding Domain and Data model	Project Software Design Specification - Team 35

Table of Contents

1 Introduction

This is Project XLR8's Network Bandwidth Tracker software's Software Design Specifications (SDS) document. In the current digital era, when connectivity is essential, it is critical for both consumers and enterprises to comprehend and optimise network bandwidth. This need is met by our software, which offers a complete solution for measuring and monitoring bandwidth consumption and internet speed. This document provides an overview of the NBT software's architectural design and technical details, as well as functionalities, design objectives, and quality of service considerations. Stakeholders will obtain a thorough understanding of the NBT system through this SDS, facilitating wise decision-making and effective implementation.

1.1 Purpose

A software system's architecture and technical specifications are described in the Software Design Specifications (SDS) document. It functions as a guide, a means of communication, and a foundation for verification. It is divided into sections for developers, testers, managers, administrators, and stakeholders, such as Use Case View, Design Goals, Data View, and Quality of Service. It is used by developers to provide implementation direction, testers to provide validation criteria, managers to track progress, administrators to provide deployment insights, and stakeholders to comprehend the capabilities of the system. SDS makes ensuring that throughout the software development lifecycle, there is consistency, clarity, and alignment with design goals throughout the software development lifecycle, aiding in maintenance, upgrades, and extensions. It outlines assumptions, constraints, and interfaces, addressing availability, security, and performance. Ultimately, SDS acts as a reference point, guiding the development, deployment, and maintenance of the software system.

1.2 Scope

The Software Design Specifications (SDS) document serves as a guiding framework for the development of the website's software. It shapes the entirety of the website's operational design, from the intricacies of data management to the user interface experience. By focusing on the project's objectives, limitations, and technological prerequisites, the SDS ensures that developers construct a system capable of accurately gauging internet speed, safeguarding data, and offering an intuitive user interface. Furthermore, it addresses critical aspects such as system performance, adaptability, and dependability, thus guaranteeing that the website not only meets but exceeds industry standards and user expectations. Essentially, the SDS forms the cornerstone upon which the entire software architecture of the website is constructed.

1.3 Definitions, Acronyms, and Abbreviations

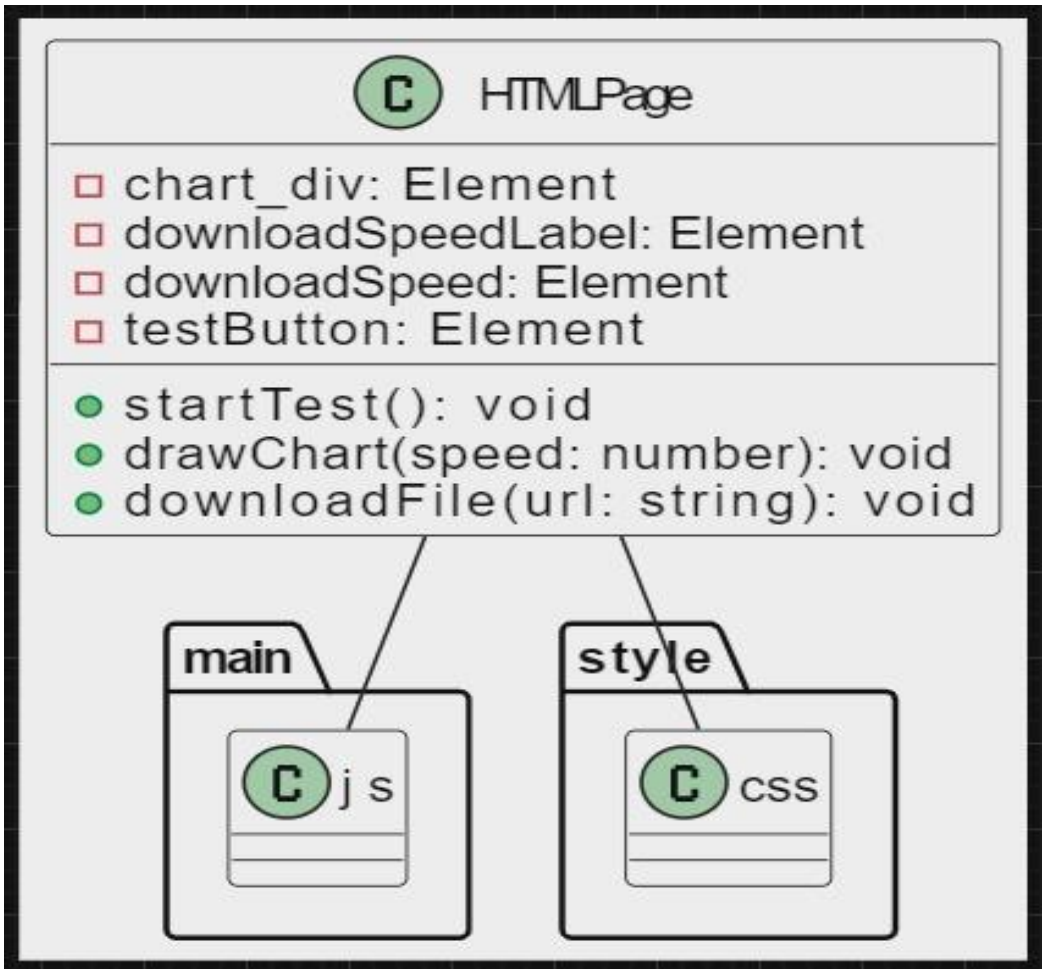
SNMP – Simple Network Management Protocol
API – Application Programming Interface
HTML – Hyper Text Markup Language
CSS – Cascading Style Sheets

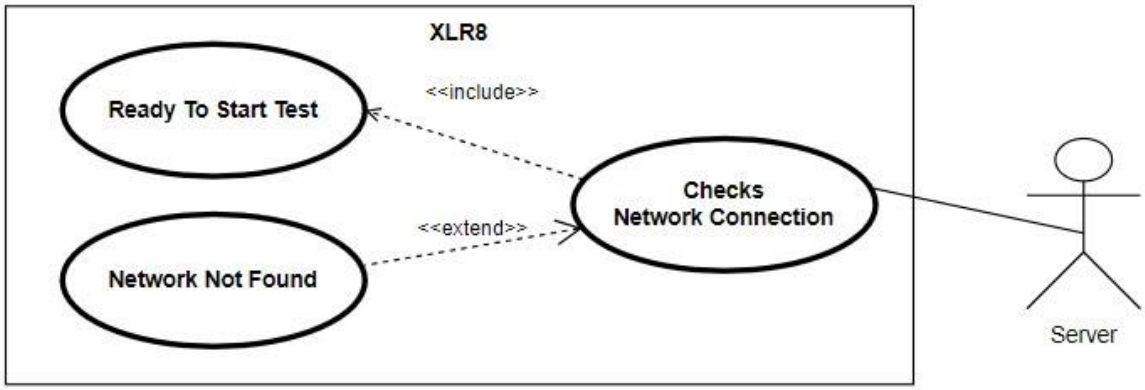
1.4 References

- Name: Design Documentation in Software Engineering
    - URL: https://www.geeksforgeeks.org/design-documentation-in-software-engineering/

2 Use Case View

2.1 Use Case

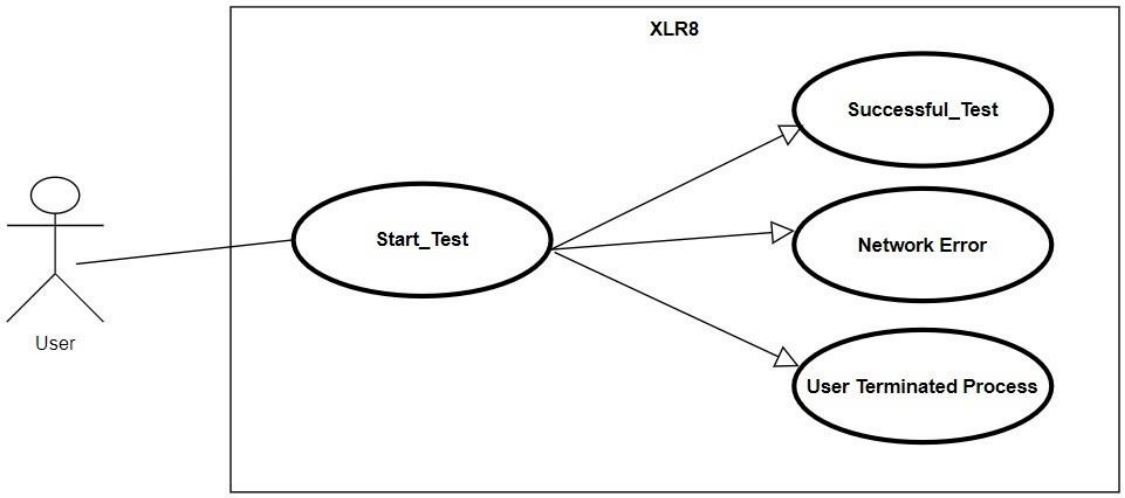




The Server/Admin performs the connectivity check required for the speed test to start and end with required results.

XLR8-NETCHECK-000:
Admin checks for a connection to be provided through the web, for a stable internet throughout the process of testing
After a successful connection, the admin signals it is ready to start the process.

XLR8-NETCHECK-001:
Admin checks for a connection to be provided through the web, for a stable internet throughout the process of testing
After an unsuccessful connection, the admin signals it is not ready to start the process, and to try again after some time.



The User gives his permission to go ahead with the speed test, using a “Start” button.

XLR8-SPEEDTEST-000: Test Ended
Admin checks for a connection through to the available server and starts the test after the user clicks the button.
After a successful test of 30 seconds, the admin displays the output in a designed form.

XLR8-SPEEDTEST-001: Unstable Internet Connection
Admin checks for a connection through to the available server and starts the test after the user clicks the button.
If the test is interrupted in between due to network issues, the admin issues an error prompting the user to come back after a few moments.

XLR8-SPEEDTEST-002: User terminated process
Admin checks for a connection through to the available server and starts the test after the user clicks the button.
If the test is interrupted in between due to the user closing the tab/window, the admin severs the connection between the server and user.

3 Design Overview

3.1 Design Goals and Constraints

Based on the provided code for an internet speed test application, here are the design goals and constraints inferred from the implementation:

1. Real-time Monitoring:
- Implement real-time monitoring of download speed to provide immediate feedback to users during speed tests.

2. User-friendly Interface:
- Design a visually appealing and intuitive interface for conducting speed tests, displaying results, and initiating test runs.

3. Accurate Speed Measurement:
- Ensure accurate measurement and representation of download speed data to provide users with reliable information about their internet connection performance.

4. Integration with Google Charts:
- Successfully integrate Google Charts library to create gauge charts for visualizing download speed data in real-time.

Constraints:

1. Browser Compatibility:
- Ensure compatibility with various web browsers to ensure consistent functionality and performance across different platforms.

2. Network Stability:
- Depend on stable network connectivity for accurate speed measurements and reliable data retrieval during speed tests.

3. Google Charts Dependency:
- Rely on external dependencies such as Google Charts library for visualization, which may introduce dependencies and potential points of failure.

4. Security Considerations:
- Address security concerns related to fetching data from external sources (e.g., file download URL) and displaying it in the web application.

5. Performance Limitations:
- Account for potential performance limitations, such as the processing power of client devices, network latency, and bandwidth restrictions, that may impact the speed test application's performance.

6. User Engagement:
- Encourage user engagement by providing clear instructions and feedback during the speed test process to ensure a positive user experience.

These design goals and constraints outline the objectives and limitations of the internet speed test application.

3.2 Design Assumptions

1. **Browser Support for Fetch API:**

Assume that the web browsers used by the target devices support the Fetch API for making HTTP requests, as demonstrated in the code for downloading files.

2. **Availability of Google Charts Library:**

Assume that the Google Charts library (loader.js) is accessible and functional, allowing the application to visualize download speed data using gauge charts.

3. **Basic Understanding of Internet Speed Testing:**

Assume that users interacting with the speed test application have a basic understanding of internet speed testing concepts, such as download speeds measured in Mbps (megabits per second).

4. **Consistent Data Retrieval from External Sources:**

Assume consistent data retrieval from external sources (e.g., file download URL) without interruptions or network errors affecting the speed test process.

5. **Sufficient Device Resources:**

Assume that the target devices running the application have sufficient hardware resources (e.g., CPU, memory) to execute the code and perform speed tests without significant performance degradation.

6. **No End-to-End Encryption for Downloaded Files:**

Assume that the files downloaded during speed tests are not encrypted end-to-end, allowing the application to accurately measure download speeds by inspecting packet headers.

7. **User Initiation of Speed Tests:**

Assume that users manually initiate speed tests by clicking the "Run Test" button on the web page, triggering the execution of the speed test logic defined in the JavaScript code.

8. **Reliable Network Connectivity for Speed Tests:**

Assume that the network connectivity during speed tests is reliable and stable, minimizing potential disruptions or fluctuations in download speed measurements

3.3 Significant Design Packages

1. **Google Charts Integration:**

Purpose: Provides visualization capabilities for displaying download speed data in real-time.

Components: Utilizes Google Charts library (loader.js) to create gauge charts for representing download speed measurements on the web page.

2. **Fetch API for HTTP Requests:**

Purpose: Facilitates the retrieval of files from external sources (e.g., file download URL) for conducting speed tests.

Components: Utilizes the Fetch API within the JavaScript code to make asynchronous HTTP requests for downloading files during speed tests.

3. **User Interface Components:**

Purpose: Renders the user interface elements for initiating speed tests and displaying download speed results.

Components: HTML markup defines the structure of the web page, including headings, buttons (<button>), and spans () for displaying download speed information.

4. **Dynamic Data Visualization:**

Purpose: Dynamically updates the gauge chart to reflect real-time changes in download speed during speed tests.

Components: JavaScript code dynamically modifies the data and options of the Google Charts gauge chart (drawChart function) based on the current download speed measurements.

5. **Asynchronous File Downloading:**

Purpose: Enables asynchronous downloading of files from external sources to measure download speeds.

Components: Utilizes asynchronous JavaScript functions and the Fetch API (downloadFile function) to initiate and monitor the progress of file downloads in real-time.

6. **User Interaction Handling:**

Purpose: Manages user interactions with the speed test application, such as initiating speed tests and displaying download speed results.

Components: JavaScript event listeners (addEventListener) attached to the "Run Test" button (testButton) trigger the execution of speed test logic (startTest function) when clicked by the user.

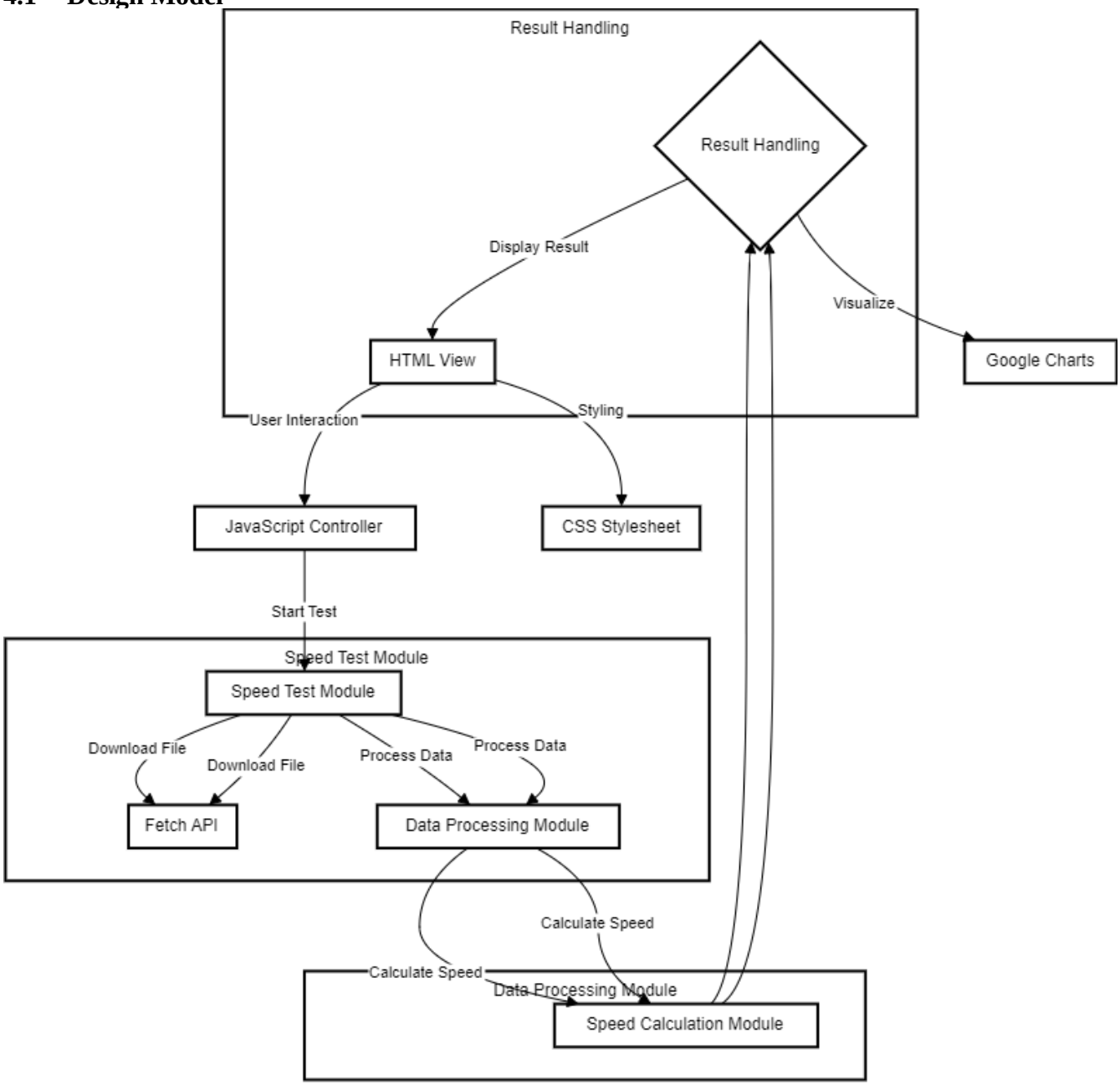
3.4 Dependent External Interfaces

The table below lists the public interfaces this design requires from other modules or applications.

External Application and Interface Name	Module Using the Interface	Functionality/
Fetch API	JavaScript code	Used for making HTTP requests. In this code, it's utilized to fetch the file from the specified URL.
Google Charts API	JavaScript code	Specifically, the corechart and gauge packages are loaded and used for drawing the gauge chart that visualizes the download speed.
SNMP (Simple Network Management Protocol)	Network monitoring module	Utilized for collecting network performance data and statistics from network devices, such as routers, switches, and servers, to provide comprehensive bandwidth monitoring and analysis
WebSocket API	JavaScript code	Utilized for establishing a persistent connection between the client and server to enable real-time data streaming, such as live updates and notifications.
Database Connection	Backend server	Interface used by the backend server to connect to the database management system (e.g., MySQL, MongoDB) for storing and retrieving data related to bandwidth monitoring, user profiles, and configuration settings.
Syslog	Logging module	Interface used for receiving and processing log messages generated by network devices, applications, and servers, providing centralized logging for monitoring and troubleshooting purposes
SNMP Trap Receiver	Network monitoring module	Interface used to receive and process SNMP trap messages sent by network devices to alert the monitoring system of specific events or conditions, such as link status changes or device failures.
Configuration File	Configuration module	Interface utilized for reading and writing configuration settings and parameters, allowing administrators to customize the behaviour and settings of the bandwidth monitoring system according to their requirements.

4 Logical View

4.1 Design Model



4.2 Use Case Realization

UseCase-1:

Sequence Diagram for Successful Connection (XLR8-NETCHECK-000):

1. Admin Module: The admin initiates the connection check process.
2. Connection Module: The system checks for a stable internet connection.
3. Connection Module: If the connection is successful, it signals readiness to the admin.
4. Admin Module: The admin receives the signal and starts the process.

Sequence Diagram for Unsuccessful Connection (XLR8-NETCHECK-001):

1. Admin Module: The admin initiates the connection check process.

2. Connection Module: The system checks for a stable internet connection.
3. Connection Module: If the connection fails, it signals the admin to try again later.

UseCase-2:

Sequence Diagram for Successful Speed Test (XLR8-SPEEDTEST-000):

1. Admin Module: The admin checks for a connection to the available server.
2. User Interface Module: The user initiates the test by clicking the button.
3. Speed Test Module: The test runs for 30 seconds.
4. Speed Test Module: If the test is successful, the admin displays the output in a designed form.

Sequence Diagram for Unstable Internet Connection (XLR8-SPEEDTEST-001):

1. Admin Module: The admin checks for a connection to the available server.
2. User Interface Module: The user initiates the test by clicking the button.
3. Speed Test Module: The test starts.
4. Speed Test Module: If the test is interrupted due to network issues, the admin issues an error prompting the user to come back after a few moments.

Sequence Diagram for User Terminated Process (XLR8-SPEEDTEST-002):

1. Admin Module: The admin checks for a connection to the available server.
2. User Interface Module: The user initiates the test by clicking the button.
3. Speed Test Module: The test starts.
4. User Interface Module: If the user closes the tab/window, the admin severs the connection between the server and user.

5 Data View

Our internet speed test application incorporates a data storage perspective to manage persistent data relevant to the system's operation. Although the application primarily focuses on real-time monitoring and analysis of internet speed, it does involve certain data storage considerations:

Download Speed Records:

The system may maintain records of download speed measurements conducted during speed tests. These records could include timestamps, download speeds, and other relevant metadata to track and analyze historical performance data.

Error Logs and Diagnostic Information:

The system might log error messages, diagnostic information, and event data to assist in troubleshooting issues, monitoring system health, and identifying potential areas for improvement.

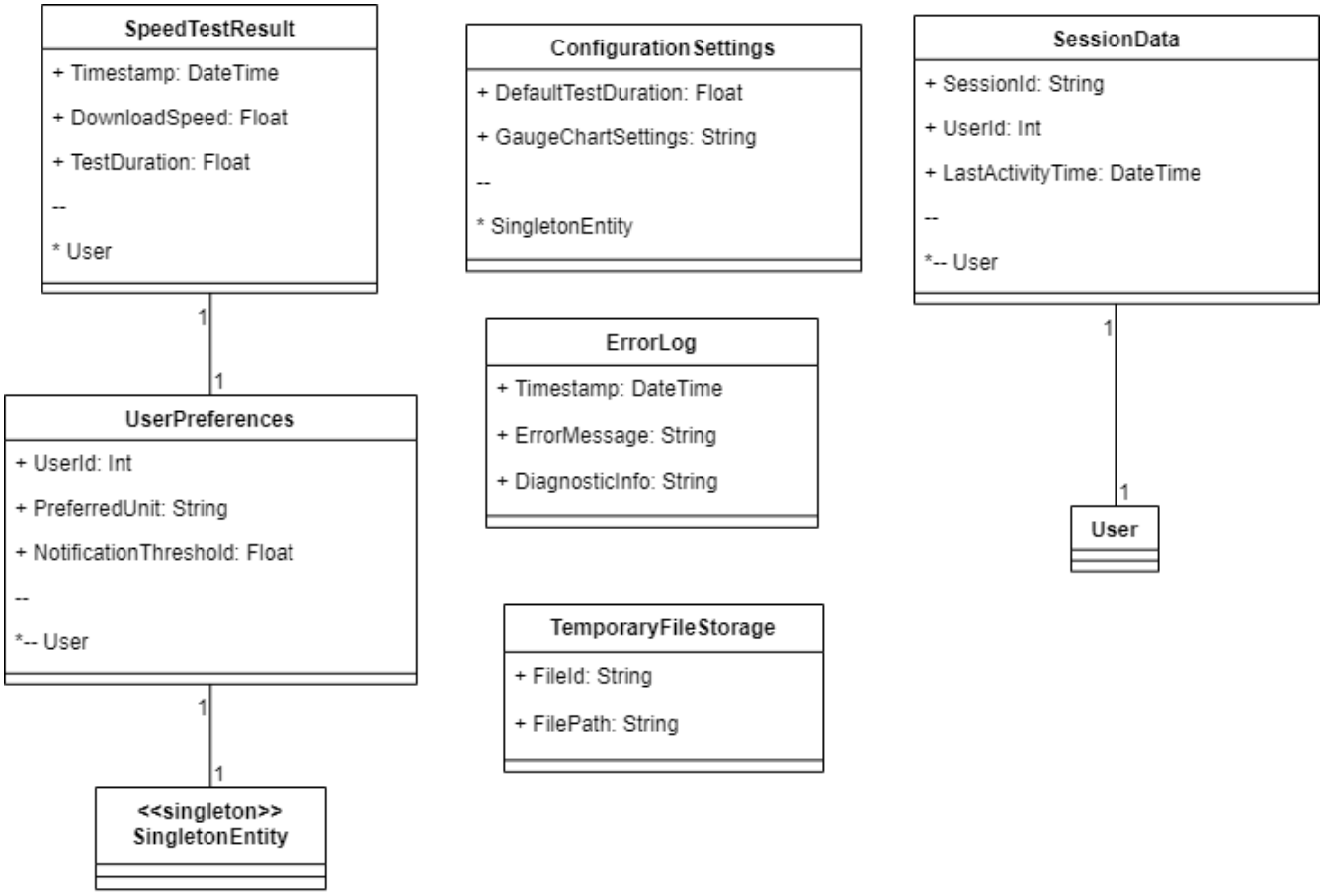
Temporary Storage for Test Files:

During speed tests, temporary storage might be utilized to buffer downloaded files before processing them for speed measurement. This temporary data storage ensures efficient handling of file downloads and speed calculations.

User Session Data:

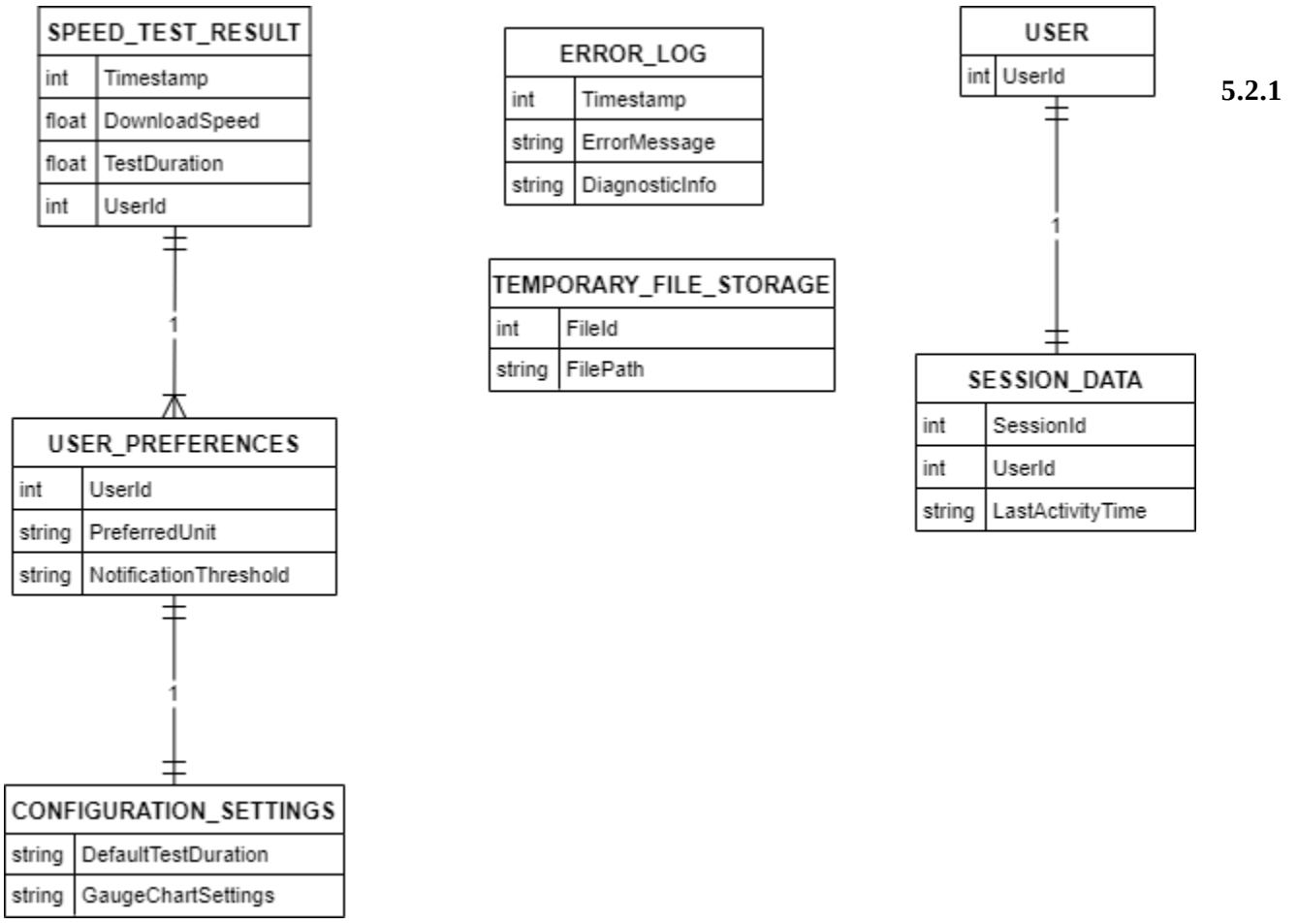
Session-related data, such as user authentication tokens, session identifiers, and user activity logs, may be stored temporarily or persistently to maintain session state and support user interactions with the application.

5.1 Domain Model



5.2 Data Model (persistent data view)

5.2.1



Data Dictionary

Entity	Attribute	Data Type	Description
SPEED_TEST_RESULT	Timestamp	Timestamp	The timestamp of the speed test result
	DownloadSpeed	Float	The download speed measured during the test
	TestDuration	Float	The duration of the speed test
	UserId	Integer	The identifier of the user who initiated the test
USER_PREFERENCES	UserId	Integer	The identifier of the user
	PreferredUnit	String	The preferred unit for displaying data
	NotificationThreshold	String	The threshold for triggering notifications
CONFIGURATION_SETTINGS	DefaultTestDuration	String	The default duration for speed tests
	GaugeChartSettings	String	Settings for the gauge chart display
ERROR_LOG	Timestamp	Timestamp	The timestamp of the error
	ErrorMessage	String	The error message
	DiagnosticInfo	String	Additional diagnostic information

SESSION_DATA	SessionId	Integer	The identifier of the session
	UserId	Integer	The identifier of the user
	LastActivityTime	String	The timestamp of the last activity in the session
TEMPORARY_FILE_STORAGE	FileId	Integer	The identifier of the file
	FilePath	String	The path to the temporary file
USER	UserId	Integer	The identifier of the user

6 Exception Handling

- **Network Errors:** Handle cases where there is a failure to establish a network connection or fetch the file for the speed test due to network issues. This could include exceptions such as FetchError or NetworkError. In such cases, log the error message and provide a user-friendly notification indicating the network problem. Prompt the user to check their internet connection and retry the speed test.

- **Timeouts:** Implement a timeout mechanism to handle cases where the file download process takes longer than expected. This could involve setting a maximum duration for the download process and throwing a TimeoutError if it exceeds this threshold. Log the timeout event and inform the user that the speed test couldn't be completed within the expected time frame. Prompt them to retry the test or check their network connection.

- **Client-Side Errors:** Address any errors that occur due to client-side issues, such as invalid configurations or incorrect user inputs. This could involve validating user inputs before initiating the speed test and throwing a ValidationError if any issues are detected. Log the validation error and provide specific instructions or feedback to the user on resolving the issue.

- **Unhandled Exceptions:** Implement a catch-all mechanism to handle any unexpected exceptions that may occur during the speed test process. Log these exceptions with detailed diagnostic information to facilitate troubleshooting and debugging. Provide a generic error message to the user indicating that an unexpected error occurred and encourage them to report the issue for further assistance.

  By incorporating these exception handling strategies into the application, we can enhance its robustness and resilience to various error conditions, providing a smoother user experience and facilitating effective troubleshooting.

7 Configurable Parameters

- **TestDuration**: This parameter defines the duration of each speed test in seconds. Users can adjust this value to customize the duration of their speed tests. Since this parameter affects the behavior of the application during runtime, it is marked as dynamic.
- **PreferredUnit**: This parameter allows users to define their preferred unit for displaying speed measurements (e.g., Mbps, KB/s). Users can modify this setting to tailor the display format according to their preferences. Since changes to this parameter can be applied without restarting the application, it is marked as dynamic.

- **NotificationThreshold**: This parameter sets the threshold for triggering speed notifications. Users can specify a threshold value, and if the measured speed falls below this threshold, a notification is generated. Since users may need to adjust this threshold based on their requirements, it is marked as dynamic.

This table describes the simple configurable parameters (name / value pairs).

Configuration Parameter Name	Definition and Usage	Dynamic?
TestDuration	Specifies the duration of each speed test in seconds	Yes
PreferredUnit	Defines the preferred unit for displaying speeds	Yes

NotificationThreshold	Sets the threshold for triggering speed notifications	Yes

8 Quality of Service

8.1 Availability

Availability Considerations:

- **Real-time Monitoring:** The software provides real-time monitoring of network bandwidth, ensuring that users have up-to-date information on their network performance.
- **Asynchronous Download:** The download process is handled asynchronously using async/await and fetch API, allowing users to continue using the application without interruption while the download test is ongoing.

Impact on Availability:

- **Network Dependency:** The software heavily relies on network connectivity for conducting download speed tests. Any network issues or outages could potentially impact the availability of the application or lead to inaccurate results.
- **Resource Consumption:** Continuous download testing may consume network bandwidth and server resources, potentially affecting the overall performance and availability of the application, especially during periods of high usage.
- **Google Charts Dependency:** The availability of the application is dependent on the availability of the Google Charts library and its associated resources. Any disruptions in the availability of Google Charts services could impact the application's ability to visualize download speed data.
- **Test Time Limit:** The application has a predefined test time limit (15 seconds in this case). If the download test exceeds this time limit, it may impact user experience and availability, especially if users expect timely results.

8.2 Security and Authorization

- **User Authentication:** Implement a login system requiring users to authenticate with a username and password before accessing the speed test features.
- **Role-Based Access Control (RBAC):** Define two roles - "admin" and "user." Administrators have access to all features, while regular users have restricted access.
- **Access Control Lists (ACLs):** Limit access to certain features or data based on user roles. For example, only administrators can access settings or view detailed test logs.

8.3 Load and Performance Implications

- **Expected Business Transaction Execution Rate:** We need to ensure that our system can smoothly handle a significant number of concurrent speed test requests. This involves optimizing resource allocation, minimizing processing time, and implementing concurrency controls to prevent any potential bottlenecks.
- **Database Table Growth Projections:** As usage of our application grows, so will the size of our database tables storing test results and user data. We must design our database schema to accommodate this growth, including implementing appropriate indexing, partitioning, and data archival strategies to ensure optimal performance as our database size increases over time.
- **Scalability and Elasticity:** Our system needs to be designed for horizontal scalability, allowing us to seamlessly handle increased load by adding more instances or resources on-the-fly. Components such as load balancers, auto-scaling groups, and distributed computing frameworks will be crucial in ensuring that we can dynamically scale our system in response to varying levels of demand.

8.4 Monitoring and Control

- **Error and Exception Rates:** Monitoring the frequency of errors and exceptions thrown by message handlers and daemons will help us identify potential issues in our application's logic or infrastructure. Tracking error rates over time will allow us to proactively address any emerging issues and ensure the stability of our system.
- **Latency and Response Times:** Measuring the latency and response times of message handlers and daemons is crucial for assessing the overall performance and responsiveness of our application. Monitoring these metrics

will help us identify any slowdowns or performance degradation and take appropriate action to optimize performance.

- **Task Completion Rates:** Tracking the rate at which tasks are completed by daemons and message handlers provides valuable insights into the efficiency and workload of our application. Monitoring task completion rates will allow us to detect any processing bottlenecks or resource constraints and make necessary adjustments to improve throughput.

- **Resource Usage Trends:** Monitoring trends in resource usage, such as CPU, memory, and disk utilization, over time will help us identify patterns and anticipate future resource requirements. This information will enable us to scale our infrastructure proactively and ensure that we can accommodate growing demand without impacting performance.