**Online real time support app for software**

This project is about an **online real-time support application** designed to connect users with technical support personnel for software-related issues. The application is built using Java and implements WebSocket technology for bidirectional real-time communication. Here’s a detailed description of the project:

**Purpose:**

The goal of the application is to allow users facing issues with specific software to connect with a support assistant who has expertise in that software. This real-time communication helps resolve issues efficiently by allowing both the user and the support assistant to exchange messages, send files, and track the progress of the support session.

**Key Components:**

1. **User & Tech Assistant Interaction:**
   * **Users** are the people facing issues with the software and require help.
   * **Tech Assistants** are employees with knowledge of specific software who provide support to the users.
2. **WebSocket Communication:** The application leverages **WebSocket** technology to maintain an open, bidirectional communication channel between the server, the users, and the tech assistants. This ensures that messages and file transfers happen in real-time without delays.
3. **Server & Client Architecture:**
   * **Server:** A WebSocket server that manages connections, ensuring that users are matched with the correct assistant based on the software they are using. The server manages messages and file transfers between users and assistants.
   * **Client (Tech Assistant & Customer):** Both the tech assistant and the user have separate client applications that connect to the server, enabling them to send and receive messages and files.
4. **User Interface:**
   * Users interact with the application by providing basic information like their username and the software they need help with. They also need to accept data privacy terms and describe the issue they are facing.
   * Tech assistants log into the system by providing their name and the software they support, making them available to assist users.
5. **Matching Process:**
   * When a user connects, the system searches for a tech assistant who is available and qualified to assist with the specific software the user is using. If a match is found, the connection is made, and the user can communicate with the assistant.
   * If no assistant is available, the system notifies the user and ends the session.
6. **Message and File Handling:**
   * Both users and assistants can exchange text messages and share files. For testing purposes, a sample file (text.txt) is available for users to send.
7. **System Flow:**
   * The user connects to the system, provides their problem description, and waits for a support assistant.
   * The system either connects the user to a suitable assistant or notifies the user if no assistant is available.
   * Once connected, the user and assistant can communicate in real-time, sending messages and files.

**Technologies Used:**

* **Java:** For the development of the server and client-side applications.
* **WebSocket:** To enable real-time, full-duplex communication between clients and the server.
* **IntelliJ IDE:** For code development.

**Features:**

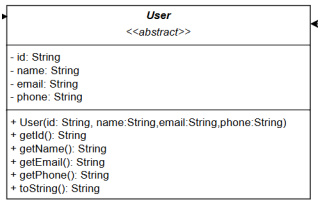
* **Real-time communication:** Users and assistants can exchange messages instantly.
* **File sharing:** Users can send files like screenshots or logs to help assist with troubleshooting.
* **Data privacy:** Users must accept terms for personal data processing before using the system.
* **Error Handling:** The system handles cases where no assistant is available, providing appropriate notifications to the user.
* **Easy-to-use interface:** The application is designed to be user-friendly, allowing both tech assistants and users to easily interact with the system.

**Application Flow:**

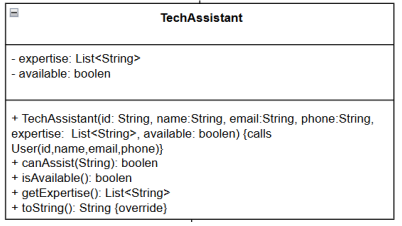
1. The tech assistant and the user connect to the server.
2. The system matches the user with an available tech assistant based on the software being used.
3. Once connected, both parties can exchange text messages or share files.
4. The session can end when either party chooses to exit the application.

Below are the classes used in the implementation of the application, along with their variables and methods:

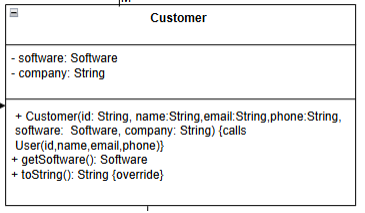
**- User class** for storing user data



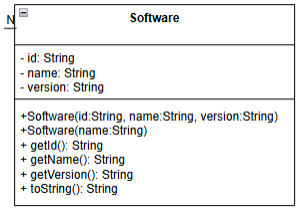
**- TechAssistant class** for storing data of the support employee (inherits from the User class)



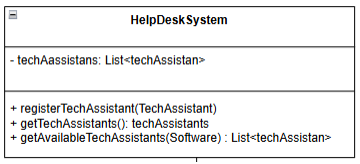
**- Customer class** for storing data of the user (inherits from the User class)



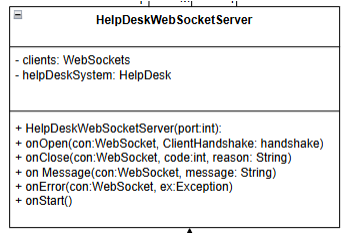
**- Software class** for storing data about the software



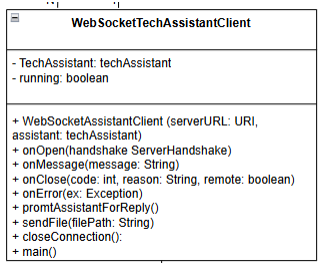
**- HelpDeskSystem class**. It provides methods to match the software used by the user with technical personnel who have knowledge of that specific software



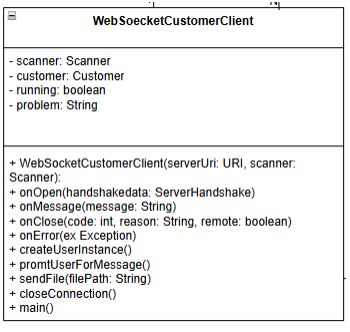
**- HelpDeskWebSocketServer class** for creating the server and managing and routing messages



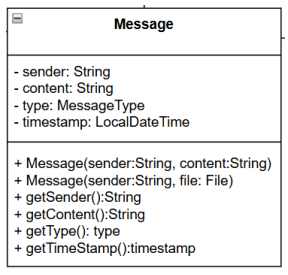
**- WebSocketTechAssistantClient class** for connecting the tech assistant to the server



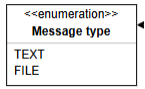
**- WebSocketCustomerClient class** for connecting the software user to the server



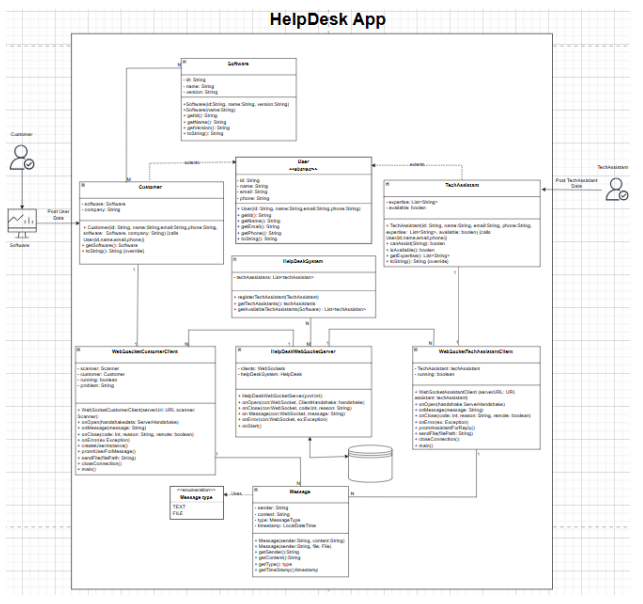
**- Message class** for message sending



- An enumeration data structure that connects to the Message class to differentiate between text messages and file sending



The design of the classes was done in draw.io (file HelpDeskApp.drawio.png), where the relationships between the classes are also shown.



**Description of the application functionality**:

The user is logged in with credentials to the software. So, we assume that the user's and the software's details are known.  
The server is running either in the cloud or on a server at the company providing support and is waiting for a user (customer or assistant) to connect.  
The assistant connects to the server by providing their details (name and software they are knowledgeable about).  
The customer connects to the server (the details of the software and the user are already known, so they are stored automatically). Additionally, the user must accept the terms for personal data processing and provide a brief description of the problem they are facing.  
The system searches for an available employee to provide support for the specific software. If an assistant is found, the connection is made; if not, it is terminated. In both cases, appropriate messages are returned to the user.  
Once the connection is made, both the user and the assistant have the ability to send messages, send files, or end the connection.

**Instructions for running the application**:

1. Run the HelpDeskWebSocketServer file to start the server.
2. Run the WebSocketTechAssistantClient file so that the tech assistant connects to the server. You will need to provide the following details:
   * the assistant's name, and
   * the name of the software that the assistant provides help with.  
     **Note**: The user in the application uses the software "AppA". Therefore, if you want to test the application behavior when there is no assistant registered in the system with expertise in the software used by the user, you should specify something other than "AppA" as the software name.
3. Run the WebSocketUserClient file to connect the user to the application. Initially, you will be asked for consent to store the data, followed by a brief description of the problem the user is facing.  
   The user’s details and the software they are using are assumed to be already known (they were recorded when the user logged into the software, with the name: UserA and software name: AppA).
4. Once the connection is established, the application checks if there is an available employee to support the "AppA" software. If no matching assistant is found, the system returns the message: "There is no available employee for help at the moment. Please try again later." If a match is found, the user is connected.
5. The system sends a message to the assistant with information about the user's details, the software, and a brief description of the problem.
6. The system notifies the user that an available employee has been found to provide assistance.
7. Whenever the user or the assistant wants to send a message or a file, they must first press "Enter". Then, the following options are available:
   * **Send a simple message**: The user simply types the message they want to send.
   * **Send a file**: The user selects "file" if they want to send a file. For testing purposes, the file "text.txt" is available.
   * **Exit the application**: The user selects "exit" to leave the application.