0. Authors A. Nerlikh, A. Maslova, S. Efimov

1. Goals and limitations

1.1. Key functional requirements The system is designed to integrate the NSUTS testing system into the VS Code development environment. The architecture is driven by the following strategic use cases:

- **Authentication:** The user must be able to log in using NSUTS credentials.
- **Contest Navigation:** The system must display a list of active contests, rounds, and tasks in a tree view.
- **Solution Submission:** The user must be able to select a compiler and submit single or multiple files (as an archive) for review.
- **Feedback Loop:** The system must display the pass report (status) and attempt history without requiring the user to switch to a browser.

1.2. Non-functional requirements

- **Platform Compatibility:** The extension must be fully compatible with VS Code version 1.104.0 and higher.
- **Network Performance:** A stable connection with a minimum speed of 1 Mbps is required.
- **External Dependency:** The extension works exclusively with the official NSUTS API; changes to this API must be reflected in the extension to maintain functionality.

1.3. Architectural goals The primary architectural goal is to eliminate the need to switch between windows (browser and IDE) during competitions.

- **Unified Context:** The system must maintain a unified task context, ensuring every action (submission, result viewing) is tied to the specifically selected task.
- **Adaptability:** The architecture must be robust enough to handle the integration with an undocumented NSUTS API and adapt to potential API changes.

1.4. Specific restrictions and constraints

- The application works as a client-side plugin within the VS Code environment.
- Authentication state must be persisted via cookies.
- For multi-file solutions, the system must automatically package files into an archive before submission.

2. Goals analysis

2.1. Security and Authentication Due to the nature of the NSUTS system, the plugin must handle user credentials securely to establish a session.

- **Session Management:** Upon successful login via login/password pair, the system receives and stores an authentication cookie. This cookie acts as the authorization token for subsequent requests (viewing tasks, submitting code).
- **Session Termination:** Logging out must clear the stored authentication cookie and terminate the session.

2.2. User Interface (VS Code Integration) The architectural challenge is to map a web-based workflow into an IDE interface.

- **Visualization**: Contests and tasks are represented as a hierarchical tree structure (Contest -> Tour -> Task) instead of traditional web pages.
- **Feedback**: Verification status (e.g., OK, Compilation Error) is displayed in a dedicated panel, updating in real-time.
- **Integration**: The **VS Code Tree View API** was chosen over a custom Webview implementation to ensure seamless native theming.

3. Solution description

3.1. Modules and subsystems The system consists of the following top-level logical modules:

- **Authentication Module:** Manages user login, logout, and secure storage of the session cookie.
- **API Client Module:** Handles all HTTP requests to the NSUTS server, including fetching contest lists and submitting solutions.
- **UI/UX Module:** Renders the contest tree view, task description panel, and submission controls within VS Code.
- **File Management Module:** Handles file selection and archiving (zipping) for multi-file solutions.

3.2. Deployment

- **Client Node:** The software is deployed as a VS Code Extension running on the user's local machine (Windows, Linux, or macOS support implied by VS Code).
- **Server Node:** The extension interacts remotely with the existing NSUTS Server via HTTP requests over the Internet.
- **Environment:** The user must have VS Code 1.104.0+ installed.

4. Key architectural elements

4.1. Communication Protocol Interaction between the VS Code plugin and the NSUTS server relies on HTTP requests.

- **Request Format:** Data transmission (e.g., task IDs, compiler selection) is handled via standard HTTP methods.
- **State Maintenance:** The session is maintained via the authentication cookie obtained during the login phase.

4.2. API Integration The system relies on an undocumented NSUTS API.

- **Constraint:** Since the API is not officially documented, the API Client module must be flexible to accommodate potential changes in the external system.
- **Functionality:** The API handles fetching contest trees, uploading solution files, and polling for results.

4.3. Data Handling

- **Submission Data:** Source code files are packaged (zipped if multiple) and sent to the server along with metadata (Task ID, Compiler ID).

5. Platform

5.1. Client Platform

- **IDE:** Visual Studio Code (version 1.104.0 or higher).
- **Languages:** The logic of the plugin is implemented using the VS Code API, which implies TypeScript.
- **Library:** Axios for requests

5.2. External Dependencies

- **Target System:** NSUTS Testing System.
- **Network:** Requires ≥1 Mbps bandwidth.