Results

Automatic Project Detection And Tooling For Devs

During the project with the Automatic Project Detection and Tooling for Devs team, the primary goal was to create a program that simplifies the process of running executable files for developers. The tool is designed to automatically detect all runnable scripts and executables within the given working directory. It not only identifies these files but also provides an intuitive interface for executing them with ease. Users can supply the necessary arguments directly through the tool, ensuring seamless and efficient script execution. This innovation significantly streamlines the workflow, saving developers time and effort while improving overall productivity.

1 Marketing

We have created a marketing poster with the team, showcasing the main pages of our software. Alongside this, we also produced a video that highlights the key features and demonstrates how to use them effectively. Both the poster and the video aim to provide a clear and engaging overview of our software, helping potential Users to understand its capabilities and benefits.

1.1 Marketing poster

The poster is available on github,

1.2 Marketing video

The video is available on github.

1.3 Software requirements

The following tools and environments are required:

- Operating System: Windows/Linux/MacOS
- Programming Language: Node.js or Python (v3.8+)
- Additional dependencies: PyQt5
- Disk space: At least 500MB of free space
- RAM: 4GB minimum (recommended 8GB for better performance)

2 Software installation

To install the software, follow these steps:

1. Open a command line prompt, where you want to install your software
2. Clone the repository: `git clone https://github.com/CsullogBeni/szofttech.git`.

Figure 1: Marketing poster

3. Activate conda environment or download Python v3.8 or higher
4. Install dependencies: Run `pip install PyQt5`
5. Set python path: `set PYTHONPATH=<Full path to the directory that contains the project>`
6. Start the application from a command line prompt. Navigate to the directory that contains the project. Run: `python ./src/view/main.py`.

3 Project tools

3.1 Version Control System

Throughout the development process, we utilized Git as our version control system and hosted the project on GitHub. The entire project including tasks, pull requests, documentation and a detailed README file is available there. You'll also find various marketing materials, brainstorming notes and ideas shared during the project. This comprehensive repository reflects both the technical and creative efforts behind our work, making it a valuable resource for anyone interested in exploring the project's details.

Useful links to the project:

- Github that contains the whole project:
  https://github.com/CsullogBeni/szofttech
- Documentation that contains the User stories, MVP, Plan, Retrospective and results:
  https://github.com/CsullogBeni/szofttech/tree/main/documentation
- Here are all the tasks throughout the project:
  https://github.com/CsullogBeni/szofttech/issues?q=
- All the pull requests that are made:
  https://github.com/CsullogBeni/szofttech/pulls?q=
- You can find here the directory that contains all the implemented source files:
  https://github.com/CsullogBeni/szofttech/tree/main/src
- We implemented unit test and system test files for the project:
  https://github.com/CsullogBeni/szofttech/tree/main/tests/unit
- Also we have created a code base that contains many runnable scripts, and we have used them to test our program:
  https://github.com/CsullogBeni/szofttech/tree/main/tests/files
- We made a few discussions as well:
  https://github.com/CsullogBeni/szofttech/discussions

4 Project team

Members:

- Zsófia Laczkó
- Borbála Merth
- Márton Petes

- Dániel Gergely
- Benedek Csüllög

4.1 Zsófia Laczkó

Full stack developer
Main tasks:

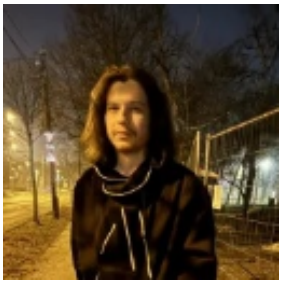
- Implementation in Model layer
- GUI implementation

4.2 Borbála Merth

Software engineer and architect
Main tasks:

- Architecture of the program
- Program documentation

4.3 Márton Petes



Test engineer and software engineer
Main tasks:

- Testing the program
- Implementation in Model layer

4.4 Dániel Gergely

Full stack developer
Main tasks:

- Implementation in Persistence layer
- GUI implementation

4.5 Benedek Csüllög

Software engineer and architect
Main tasks:

- Architecture of the program
- Implementation in Model layer

5 Completed MVP parts

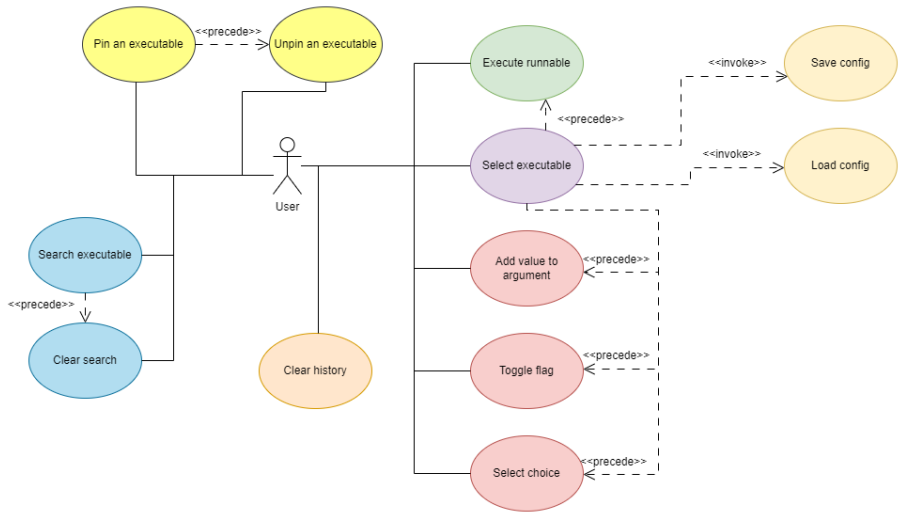


Figure 2: Use case diagram

5.1 Content of the MVP

Here are the MVP points that we have fulfilled:

- Select executable
- Add values to arguments
- Toggle flag
- Select choices
- Execute program
- Save config
- Load config
- Clear history
- Search executable
- Clear search
- Pin an executable
- Unpin an executable

5.2 MVP parts in the program

After the program initialization, on the main page:

- The Users are able to set the current working directory.
- Users are also able to set their configuration history.
- The Users can search for their executable scripts from the working directory.
- They can also clear their search.
- On the main page all the pinned executable files are shown first.
- After the pinned ones, there are all the other scripts.
- Next to the executable button the Users can set the pinned status of a script.

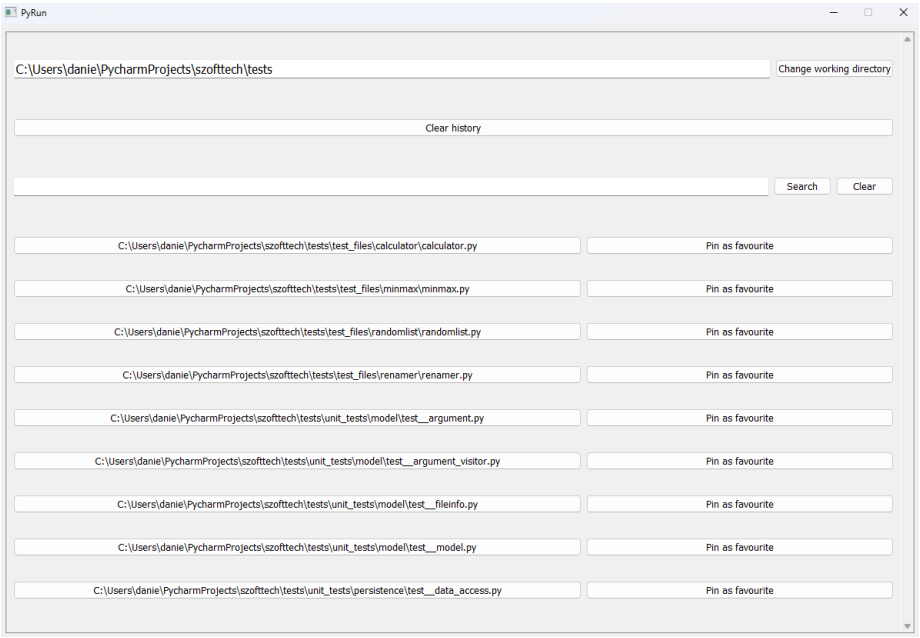


Figure 3: Main page

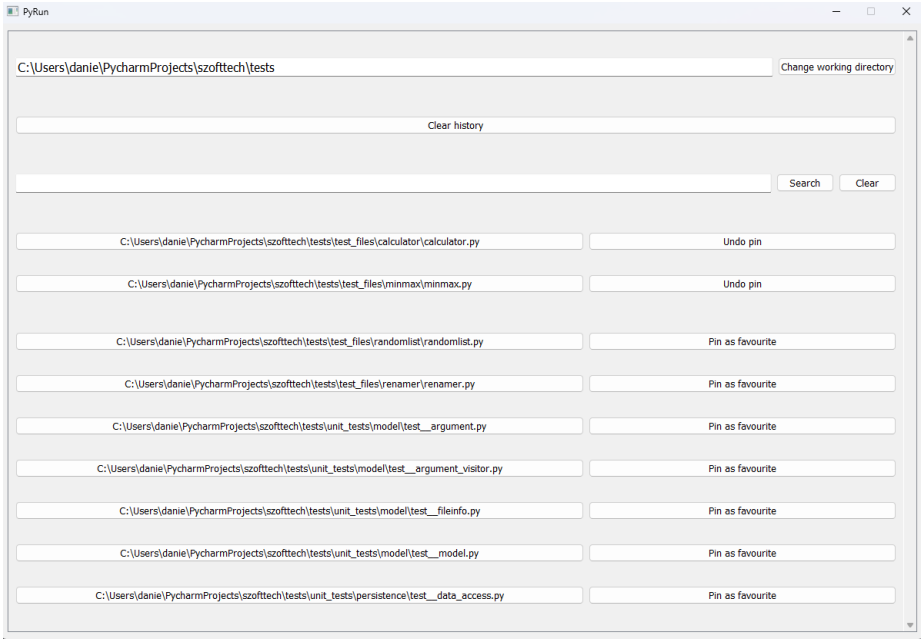


Figure 4: Main page with pinned runnable

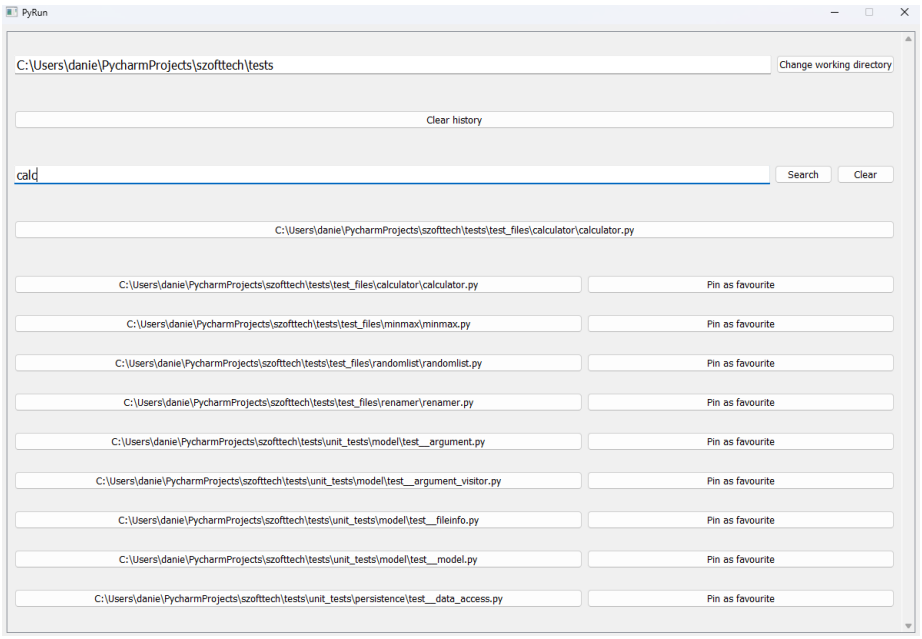


Figure 5: Main page when the User searches

After the User clicks on an executable, on the configuration page:

- The Users can see every important information about the executable.
- The Users are able to add values to the arguments in an input field.
- They can toggle flag arguments of the executable.
- If the argument contains choices, the program offers these choices in a list format, and the Users can select from the given list.
- The program saves the last configuration automatically and if the same configuration page is opened again, the program loads the last argument configuration.
- One of the main feature of the program is that the Users can execute the scripts directly from the tool.
- Last but not least, the Users can clear the current configuration.

Figure 6: Configuration page

After running a script the Users are able to see the output of a program, or the error message if something goes wrong.

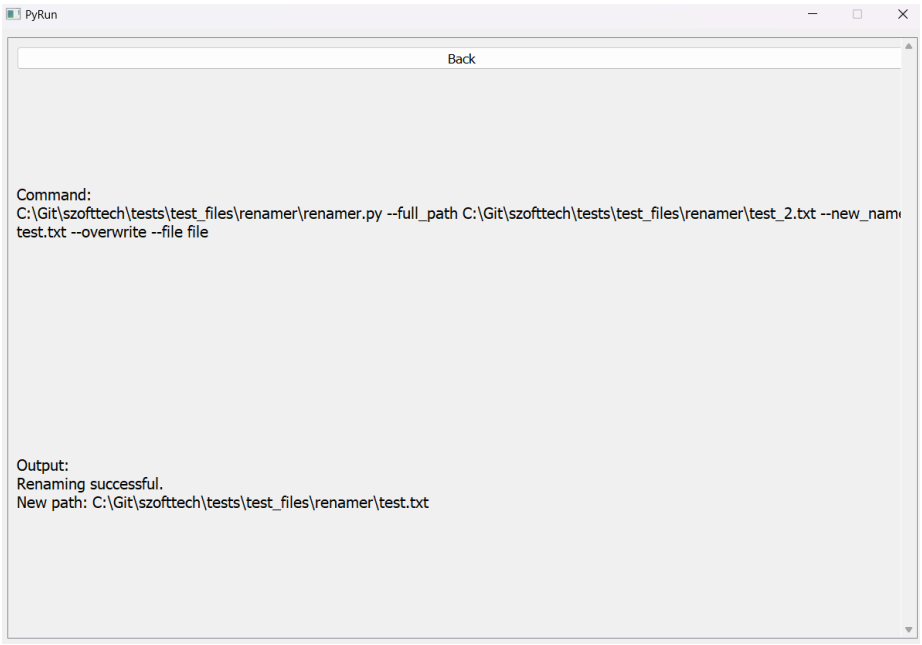


Figure 7: Runner page

5.3 Conclusion

We are pleased to confirm that all the MVP features outlined above have been successfully implemented in the program. Each point has been carefully developed and integrated, ensuring the application meets the initial goals and requirements.

In conclusion, we have tried to create a program that simplifies developers' lives by addressing common challenges and streamlining their workflow. While the program already offers valuable features, it holds significant potential for further development. Enhancements could make it even more universal, enabling it to cater to a broader range of needs and Users. By continuing to refine and expand its capabilities, the program could become an indispensable tool for developers worldwide.

5.4 Ideas for the future

Points that are not in the MVP, but in future iterations can be new features:

- .exe for the program - easier program start

- Our software could potentially recognize, analyze, and execute executable scripts written in multiple different programming languages. Each new programming language would be an extension of the product.

6 User Statistics

The completed software was tested by our IT colleagues. The goal was to assess the program's functionality, usability, and performance in real-world scenarios.

6.1 Participant Demographics

Number of Participants: 5
Professional Background: All participants have an IT or software development background.
Roles Involved:

- DevOps Engineer: 1
- Front-end Developer: 1
- Back-end Developer: 1
- Full-stack Developer: 1
- System Administrators: 1

6.2 Methodology

Duration: Each participant tested the program for approximately 2 hours.
Environment: Testing was conducted on individual workstations with different hardware and software configurations.
Test Scenarios: Participants were asked to perform the following tasks:

1. Load predefined scripts.
2. Handle the loaded scripts (search, pin as favourite).
3. Configure different arguments of the scripts.
4. Execute scripts.
5. View the outputs after successful and unsuccessful runs.

6.3 Key Metrics Collected

Task Completion Rate:
Most participants successfully completed all assigned tasks within the given time frame.

- Overall: 100%
- Loading and Executing Scripts: 98%
- Error Handling and Log Viewing: 96%

Average Task Completion Time:

- Loading and handling scripts: 5 minutes
- Execute scripts: 15 minutes
- Analyse outputs 8 minutes

Error Rate: 5%, common issues identified:

- Script syntax validation: 30%
- Display of outputs: 20%
- External dependency management: 45%
- Other: 5%

Performance Metrics:

- Script Execution Success Rate: 98%
- Average Script Execution Time:
  - Small scripts (¡100 lines): 1.5 seconds
  - Medium scripts (100-500 lines): 3.2 seconds
  - Large scripts (¿500 lines): 7.8 seconds
- Peak Memory Usage: our program has low memory usage, so it depends on the scripts

6.4 User Feedback Summary

- Ease of Use:
  95% of participants found the GUI intuitive and user-friendly. Suggested improvements included adding tooltips.
- Performance:
  The majority reported smooth performance. One participant noted a slight delay when handling lots of scripts.
- Functionality:
  We have got positive feedback on the implemented features.
- Stability:
  No crashes were reported during testing. Minor bugs were identified in the log display feature, which have been noted for fixing.

7 KPI goals

We have defined the KPI this way:
The KPI measures the completion rate of the core features and functionalities planned for the project. This metric helped us objectively assess the progress and success of the project by focusing on the implementation of key deliverables that form the foundation of the system's functionality.
The KPI is calculated as the percentage of core features implemented out of the total planned features. In our case, these core features were clearly defined and broken down into 12 specific use cases, each representing a critical aspect of the project's functionality. These use cases were identified during the planning phase, with careful consideration of both stakeholder requirements and technical feasibility, ensuring that they accurately reflect the project's objectives.
At the conclusion of the implementation phase, we verified that all 12 planned use cases had been successfully developed. Therefore our KPI can be calculated as follows:

$$\frac{implemented\ use\ cases}{planned\ use\ cases} * 100 = \frac{12}{12} * 100 = 100$$

As a result, our KPI is 100%, and since our target was also to achieve 100%, we can confidently state that we have successfully completed our goal. This accomplishment is a testament to the team's hard work, effective planning, and strong collaboration throughout the projects lifecycle.