Plan

Automatic Project Detection And Tooling For Devs

1 Full work breakdown

The work breakdown structure covers all the required functionality in the MVP through three milestones, each addressing core parts of the Model, Persistence, and View layers. It also contains tasks for unit testing to ensure that the program is safe to use.

Milestone 1 - Base Implementation (Due: 2024-11-10)

- Model Layer:
  - Implement base classes: Argument that represents an argument of a script, ArgumentVisitor that collects the arguments, FileInfo that contains all the data about a script.
  - Implement must-have functionalities in the Model class: program findig, program running, IO operations.
- Persistence Layer:
  - Implement IDataAccess interface and DataAccess class to be able to save and load configuration data.
- Testing:
  - Define and implement unit tests for all classes and functions in Model and Persistence layers.

Milestone 2 - View Implementation (Due: 2024-11-30)

- View Layer:
  - Implement all the widgets that are needed for each screen: text labels, buttons, combo boxes.
  - Implement all required screens to create the UI: RunnerScreen to be able to run the scripts, RunnableConfigScreen to be able to configure the scripts, ShowRunnablesScreen to be able to list all the scripts.
- Core Execution:
  - Implement main.py to integrate all screens and core functionality.

Milestone 3 - Enhanced Functionality and Testing (Due: 2024-12-14)

- Model and View Layers:
  - Implement search functionality in both Model and View.
  - Implement filtering for main runnables.
- Testing:
  - Develop a comprehensive runner for all test cases.

2 Tasks

We defined the tasks to be easy to understand and small enough to be done within a few days. The tasks must meet some criteria before getting accepted and merged into the solution:

- Unit tests must be written and passed for each new class/function.
- UI components must be validated for usability and performance.
- Core integrations should function without errors.
- Each class, interface, function etc. must have a Python docstring documentation.

The full task list is given in Task assignment (section 4).

3 Structure of tasks

Tasks are structured to reflect dependencies and flow between components:

- Milestone 1 tasks are foundational for Model and Persistence layers.
- Milestone 2 builds upon Milestone 1, focusing on the View layer.
- Milestone 3 completes advanced features and ensures all functionality is tested.

4 Task assignment

The following tables shows task assignments for each milestone to ensure accountability. Each task is assigned to at least one of the group members and all the tasks have a due date to ensure the comletion of the milestone.

4.1 Milestone 1

Task	Developer	Due Date
Implement Argument class in Model layer	Zsófia Laczkó	2024-10-30
Implement ArgumentVisitor class in Model layer	Benedek Csüllög	2024-10-30
Implement FileInfo class in Model layer	Borbála Merth	2024-10-30
Implement IDataAccess interface in Persistence layer	Dániel Gergely	2024-10-30
Implement DataAccess class inheriting IDataAccess in Persistence layer	Dániel Gergely	2024-10-30
Implement Model class's base structure in Model layer	Márton Petes	2024-11-05
Implement Model class's program finding and executable runner functions in Model layer	Zsófia Laczkó	2024-11-10
Implement Model class's IO operations in Model layer	Márton Petes	2024-11-10
Define unit tests for Argument class in Tests layer	Borbála Merth	2024-11-10
Define unit tests for ArgumentVisitor class in Tests layer	Benedek Csüllög	2024-11-10
Define unit tests for FileInfo class in Tests layer	Borbála Merth	2024-11-10
Define unit tests for Model class in Tests layer	Márton Petes	2024-11-10
Define unit tests for DataAccess class in Tests layer	Dániel Gergely	2024-11-10
Data access working directory addition	Zsófia Laczkó	2024-10-30

4.2 Milestone 2

Task	Developer	Due Date
Implement TitleTextLabel widget	Borbála Merth	2024-11-20
Implement NormalTextLineEdit widget	Borbála Merth	2024-11-20
Implement NormalTextComboBox widget	Borbála Merth	2024-11-20
Implement NormalTextLabel widget	Zsófia Laczkó	2024-11-20
Implement NormalTextButton widget	Benedek Csüllög	2024-11-20
Implement ComboBox widget in RunnableConfigScreen	Benedek Csüllög	2024-11-20
Implement RunnableConfigScreen's base structure	Zsófia Laczkó	2024-11-20
Implement RunnableConfigScreen's string operations	Zsófia Laczkó	2024-11-29
Implement RunnableConfigScreen's input field addition	Zsófia Laczkó	2024-11-29
Implement RunnableConfigScreen's equip button and its functionality	Zsófia Laczkó	2024-11-29
Implement ShowRunnablesScreen's base structure	Dániel Gergely	2024-11-29
Implement RunnableConfigScreen save/load config and clear functions	Benedek Csüllög	2024-11-29
Implement RunnerScreen	Zsófia Laczkó	2024-11-29
Implement the clear function in ShowRunnablesScreen	Márton Petes	2024-11-29
Implement main.py that integrates all the views	Márton Petes	2024-11-30

4.3 Milestone 3

Task	Developer	Due Date
Implement ShowRunnablesScreen's search field	Dániel Gergely	2024-12-14
Implement search functionality in Model class	Márton Petes	2024-12-14
Implement ShowRunnablesScreen's main indications	Benedek Csüllög	2024-12-14
Implement filtering for main runnables in Model class	Benedek Csüllög	2024-12-14
Develop test runner	Dániel Gergely	2024-12-14
Implement some test runnables	Benedek Csüllög, Márton Petes, Zsófia Laczkó, Dániel Gergely, Borbála Merth	2024-12-14

5 Time management

The timeline aligns with each milestone's due date, ensuring the project is completed on schedule:

- Milestone 1: Complete by 2024-11-10.
- Milestone 2: Complete by 2024-11-30.
- Milestone 3: Complete by 2024-12-14.
- Sufficient time is allocated for testing and integration after each milestone.

6 Architecture

6.1 Software components

The program is built on 3 main layers, each representing a namespace. These are:

- Model: handles the business logic. Finds runnables, and collects their details. Executes the runnables with given configurations.
- Persistence: handles the IO operations. Saves and loads user preferences and other information that is needed for better usability.
- View: stands for the user interface and user experience.

The 1. image shows the structure of the 3 layers, and all the files that are included.

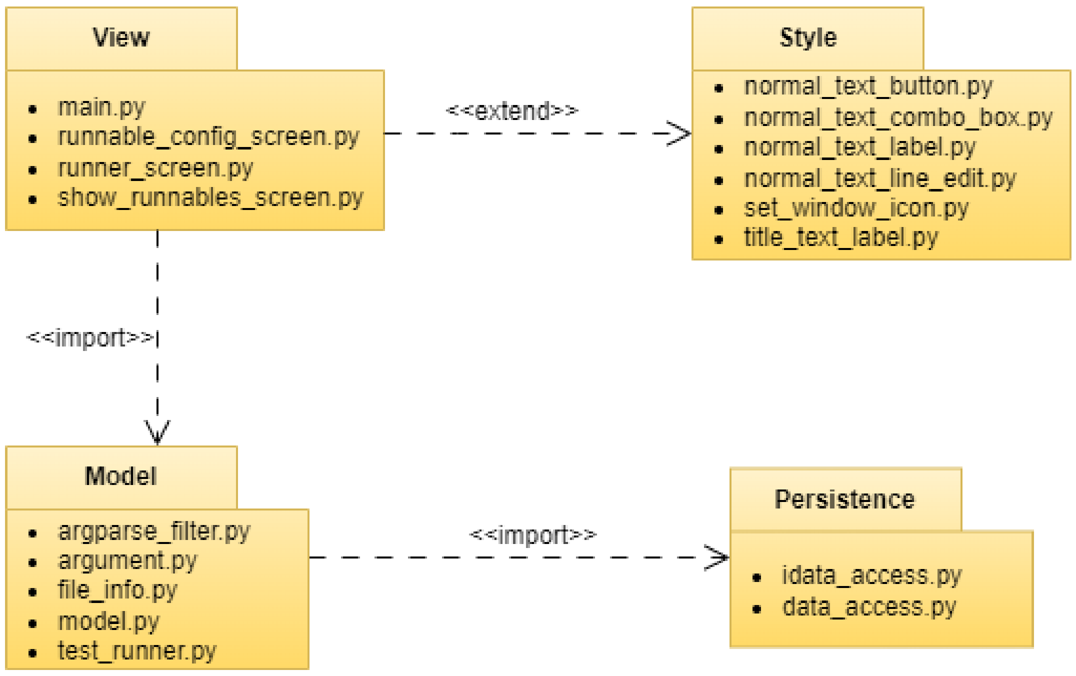

Figure 1: UML package diagram

The UML class diagram of Model and Persistence layers (2. image) shows the structure of each object within these namespaces and the relations between them.

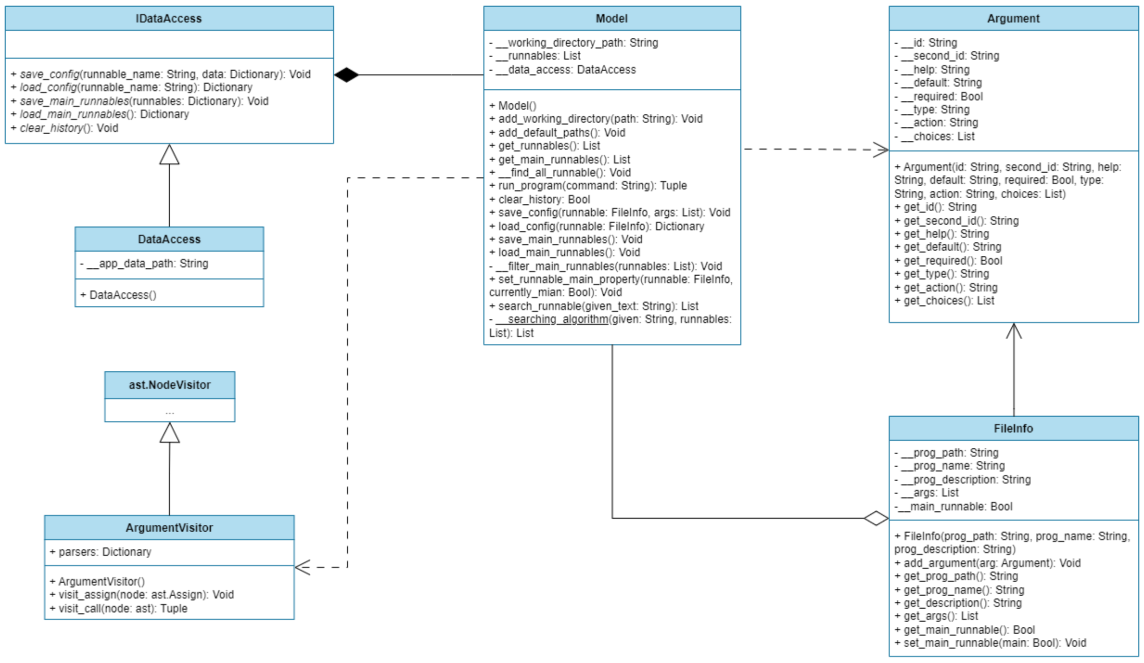
IDataAccess
+ save_config(runnable_name: String, data: Dictionary): Void
+ load_config(runnable_name: String): Dictionary
+ save_main_runnables(runnables: Dictionary): Void
+ load_main_runnables(): Dictionary
+ clear_history(): Void
DataAccess
- __app_data_path: String
+ DataAccess()
ast.NodeVisitor
...
ArgumentVisitor
+ parsers: Dictionary
+ ArgumentVisitor()
+ visit_assign(node: ast.Assign): Void
+ visit_call(node: ast): Tuple
Model
- __working_directory_path: String
- __runnables: List
- __data_access: DataAccess
+ Model()
+ add_working_directory(path: String): Void
+ add_default_paths(): Void
+ get_runnables(): List
+ get_main_runnables(): List
+ __find_all_runnable(): Void
+ run_program(command: String): Tuple
+ clear_history: Bool
+ save_config(runnable: FileInfo, args: List): Void
+ load_config(runnable: FileInfo): Dictionary
+ save_main_runnables(): Void
+ load_main_runnables(): Void
- __filter_main_runnables(runnables: List): Void
+ set_runnable_main_property(runnable: FileInfo, currently_mian: Bool): Void
+ search_runnable(given_text: String): List
- __searching_algorithm(given: String, runnables: List): List
Argument
- __id: String
- __second_id: String
- __help: String
- __default: String
- __required: Bool
- __type: String
- __action: String
- __choices: List
+ Argument(id: String, second_id: String, help: String, default: String, required: Bool, type: String, action: String, choices: List)
+ get_id(): String
+ get_second_id(): String
+ get_help(): String
+ get_default(): String
+ get_required(): Bool
+ get_type(): String
+ get_action(): String
+ get_choices(): List
FileInfo
- __prog_path: String
- __prog_name: String
- __prog_description: String
- __args: List
- __main_runnable: Bool
+ FileInfo(prog_path: String, prog_name: String, prog_description: String)
+ add_argument(arg: Argument): Void
+ get_prog_path(): String
+ get_prog_name(): String
+ get_description(): String
+ get_args(): List
+ get_main_runnable(): Bool
+ set_main_runnable(main: Bool): Void

Figure 2: UML class diagram of Model and Persistence layers

The UML class diagram of View layer (3. image) shows the structure of each object within this namespace and the relations between them.

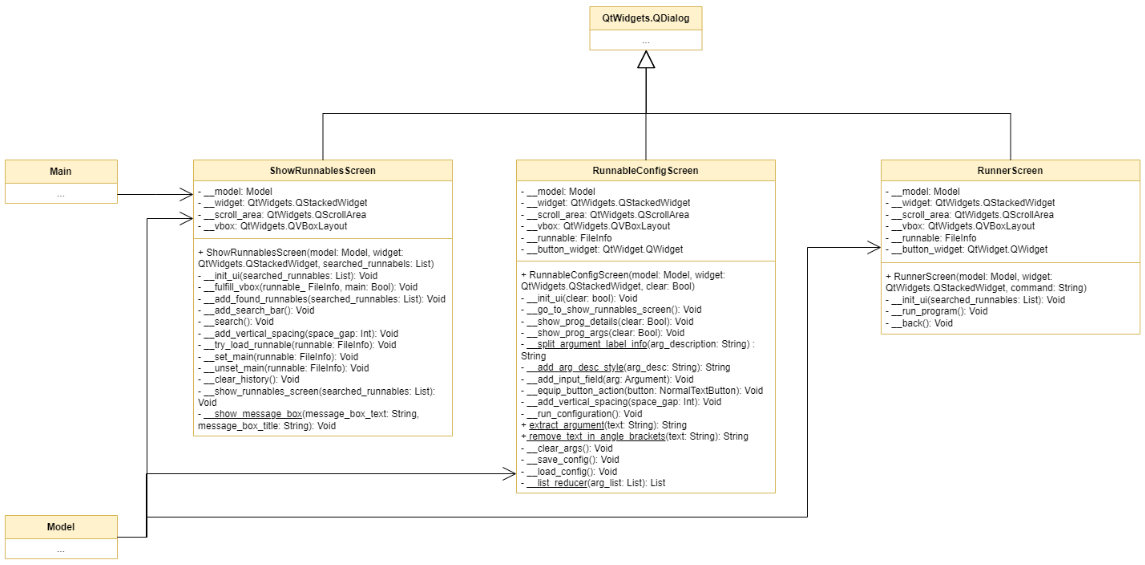
QtWidgets.QDialog
...
Main
...
Model
...
ShowRunnablesScreen
- __model: Model
- __widget: QtWidgets.QStackedWidget
- __scroll_area: QtWidgets.QScrollArea
- __vbox: QtWidgets.QVBoxLayout
+ ShowRunnablesScreen(model: Model, widget: QtWidgets.QStackedWidget, searched_runnabels: List)
- __init_ui(searched_runnables: List): Void
- __fulfill_vbox(runnable_ FileInfo, main: Bool): Void
- __add_found_runnables(searched_runnables: List): Void
- __add_search_bar(): Void
- __search(): Void
- __add_vertical_spacing(space_gap: Int): Void
- __try_load_runnable(runnable: FileInfo): Void
- __set_main(runnable: FileInfo): Void
- __unset_main(runnable: FileInfo): Void
- __clear_history(): Void
- __show_runnables_screen(searched_runnables: List): Void
- __show_message_box(message_box_text: String, message_box_title: String): Void
RunnableConfigScreen
- __model: Model
- __widget: QtWidgets.QStackedWidget
- __scroll_area: QtWidgets.QScrollArea
- __vbox: QtWidgets.QVBoxLayout
- __runnable: FileInfo
- __button_widget: QtWidget.QWidget
+ RunnableConfigScreen(model: Model, widget: QtWidgets.QStackedWidget, clear: Bool)
- __init_ui(clear: bool): Void
- __go_to_show_runnables_screen(): Void
- __show_prog_details(clear: Bool): Void
- __show_prog_args(clear: Bool): Void
- __split_argument_label_info(arg_description: String) : String
- __add_arg_desc_style(arg_desc: String): String
- __add_input_field(arg: Argument): Void
- __equip_button_action(button: NormalTextButton): Void
- __add_vertical_spacing(space_gap: Int): Void
- __run_configuration(): Void
+ extract_argument(text: String): String
+ remove_text_in_angle_brackets(text: String): String
- __clear_args(): Void
- __save_config(): Void
- __load_config(): Void
- __list_reducer(arg_list: List): List
RunnerScreen
- __model: Model
- __widget: QtWidgets.QStackedWidget
- __scroll_area: QtWidgets.QScrollArea
- __vbox: QtWidgets.QVBoxLayout
- __runnable: FileInfo
- __button_widget: QtWidget.QWidget
+ RunnerScreen(model: Model, widget: QtWidgets.QStackedWidget, command: String)
- __init_ui(searched_runnables: List): Void
- __run_program(): Void
- __back(): Void

Figure 3: UML class diagram of View layer

The View has 3 screens:

- The first one shows all the executables. An executable can be reached with a button. Every executable can be pinned as favourite. The screen also contains a search bar and a clear history button.
- The second screen shows an executable all information. Lists all the arguments and offers an opportunity to give a value to it. The user can also run the program here.
- The third and last screen shows the output messages, logs or errors for the user, after a program execution.

The screen's wireframe plan can be seen on the 4. image.

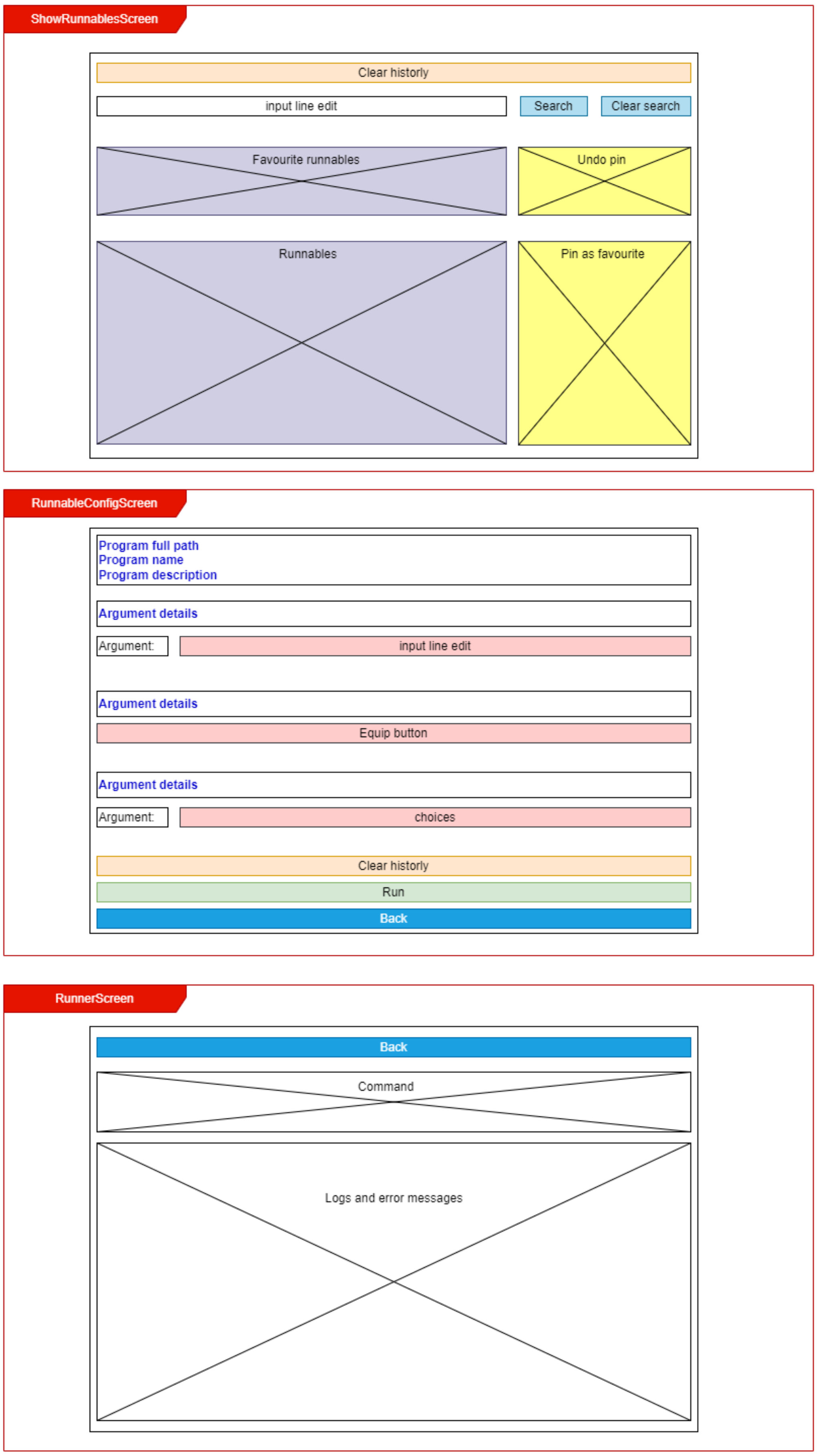

Figure 4: Screens' wire frame plan

6.2 Software installation

To install the software, follow these steps:

1. Clone the repository: `git clone https://github.com/CsullogBeni/szofttech.git`.
2. Install dependencies: Run `pip install PyQt5`
3. Set python path: `set PYTHONPATH=<Full path to the directory that contains the project>`
4. Start the application: `python ./src/view/main.py`.

6.3 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)