Universal WebParser

Purpose

Universal Web Parser is a Software-as-a-Service (SaaS) platform that lets you parse any web page using Kotlin runtime scripts, executed at scheduled intervals. Each script has access to standard Kotlin library, and helper APIs.

The application:

- manages the browser lifecycle, allowing scripts to focus purely on web scraping logic. Each script runs in an isolated browser context with its own persistent state and cookies, which are automatically saved and reloaded on every run. This allows scripts to behave more like real users - maintaining sessions, staying logged in, and avoiding detection as bots.
- uses Kotlin coroutines for efficient, scalable concurrency, with an internal scheduler, task queue, and configurable workers for parallel execution.

A 5-minute cooldown period ensures that each script runs no more than once within that window. The exact execution time depends on worker availability and the number of scripts in the queue. Each script can store up to preset number of unique results (default is 100).

User Interface

The application includes a web-based interface, providing a dashboard where you can:

- Add, edit, and remove scripts
- View compilation status, errors, and scheduled execution times
- Enable, disable, or terminate running scripts
- Inspect execution results for each script

A standout feature of the user interface is its real-time updates:

- Running scripts are highlighted in green, and all metrics update in real time
- Web console displays real-time log updates from both the application and the scripts. Messages printed from scripts are prefixed with SN>, where N is the script ID, for example:
  18:10:41 Script 2 enabled
  18:10:41 Starting script 2
  18:10:41 S2> Hello from script 2
  18:10:41 Result for script 2 is not unique, only timestamp updated

Script API

Each script is identified by a unique ID and has access to the following API:

- **println(Any)** - outputs any object to the web console
- **page** - a ready-to-use Playwright Page instance.
- **store** - provides access to state. Scripts can use it as an internal store for sensitive information or as a lightweight text storage. The store provides the following methods:
  - ***load()***: ***Map<String, String>***
  - ***save(map: Map<String, String>)***
  - ***remove(vararg keys: String)***
  - ***canProcessAfter(url: String, hours: Int): Boolean*** - Returns `true` on the first invocation for a URL, then `false` until the specified time has passed. Useful to throttle parsing. Internal storage for this method is not persisted.

The store can also be modified via REST endpoints by the admin.

Any value returned from the script is automatically converted to a JSON object; no manual conversion is needed.

Users

The project was built for personal use, and no user registration is available. However, it defines two built-in users:

- **admin** – full access to all features
- **guest** – read-only access (except for scripts store, which are completely restricted)

Passwords for both users should be configured via environment variables on deployment.

Technical Details

Structure

The project consists of the **root project** and 3 modules

- **script-definition** - defines the template class for the runtime scripts
- **script-host** - compiles and executes runtime scripts
- **script-playwright** - provides playwright dependencies to all modules and the root project

Compilation

For compilation, use ***./gradlew clean bootJar*** command.

Env Variables

- ***PLAYWRIGHT_WS***="ss://localhost:3000" # Mandatory
- ***ADMIN_PASS***=admin # Mandatory
- ***GUEST_PASS***= # Mandatory. Auto-generated if empty
- ***JWT_SECRET***= # Mandatory. Auto-generated if empty.
- ***FAIL_THRESHOLD***=3 # Optional. If present, should have value
- ***WORKERS***=1 # Optional. If present, should have value
- ***RESULT_LIMIT***=100 # Optional. Max stored results per script
- ***SPRING_PROFILES_ACTIVE***=dev # Optional. Disables SSL
- ***PORT***=443 # Mandatory if not dev profile
- ***KEY_STORE***="filename.p12" # Mandatory if not dev profile
- ***KEY_STORE_PSW***=keystorePass # Mandatory if not dev profile
- ***KEY_STORE_ALIAS***=keystoreAlias # Mandatory if not dev profile

Default profile is prod that requires ssl configuration. You can set it to dev to use plain http if you host it on local machine. Docker image for playwright server can be found here.

REST Endpoints

Results API

Returns a single result: ***GET /api/results/{resultId}***

```
{
  "scriptId": 1,
  "resultId": 42,
  "scriptName": "Example Script",
  "timestamp": "2025-11-09T12:34:56Z",
  "resultHash": "abc123",
  "status": "SUCCESS",
  "result": "any value returned from the script as json object"
}
```

Returns result for a group ***POST /api/results/group***
Accepts: **{ group: "groupName", delivered: boolean? }**
If delievered is null, returns both, delivered and undelivered results
Returns:

```
[{
    "scriptId": 2,
    "resultId": 6,
    "scriptName": "2",
    "timestamp": "2025-11-09T20:28:42.707282600",
    "resultHash": "d4735e51f90da3a666eec13ab35",
    "status": "SUCCESS",
    "result": "2",
    "delivered": true
}]
```

Marks results as delivered: ***PATCH /api/results/delivered***
Accepts an array of result IDs to mark as delivered: [42, 43, 44]

Scripts API
Returns a list of scripts: ***GET /api/scripts***

```
[{
   "id": 1,
   "name": "Example Script",
   "group": "default",
   "compiles": true,
   "runs": 10,
   "fails": 1,
   "nextRun": "2025-11-09T13:00:00Z",
   "isEnabled": true,
   "isEnqueued": false
 }]
```

Creates new script: ***POST /api/scripts***
Accepts:

```
{
  "name": "New Script",
  "group": "default",
  "source": "println(\"Hello\")"
}
```

Returns: **{ "id": 3 }**

Updates script ***PATCH /api/scripts/{id}***
Accepts:

```
{
  "name": "Updated Script",
  "group": "default",
  "source": "println(\"Updated\")"
}
```

Returns script GET /api/scripts/{id}
Returns:

```
{
  "scriptId": 3,
  "source": "println(\"Hello\")",
  "name": "New Script",
  "group": "default"
}
```

Deletes script *DELETE /api/scripts/{id}*
(And all its results along with browser profile and secrets!)

All results for script *GET /api/scripts/{id}/results*

```
[{
   "scriptId": 3,
   "resultId": 42,
   "timestamp": "2025-11-09T12:34:56Z",
   "status": "SUCCESS",
   "delivered": false
}]
```

Returns compilation logs as a raw string: *GET /api/scripts/{id}/clogs*
(Compilation logs available only for not compiling scripts)

Change script state: *POST /api/scripts/{id}/actions*
Accepts:

```
{ "action": "ENABLE"   // or "DISABLE", "FORCE_STOP" }
```

Enable makes the script available for schedule, disable does the opposite. Force-stop is used to kill running script, the script's source code should support this feature.

Save secret for script's store *PUT /api/scripts/{id}/secret*
Accepts:

```
[{ "key1": "value1", "keyN": "valueN" }]
```

Retrieve secrets from script's store *GET /api/scripts/{id}/secret*
Returns:

```
[{ "key1": "value1", "keyN": "valueN" }]
```

Killing Scripts
A running scripts can be terminated (cancelled) at any time using FORCE_STOP on actions endpoint, it also marks script as disabled automatically, to prevent its rescheduling. A script can only be killed if it:

- Uses Kotlin co-routines and have suspension points
- Checks ***Thread.currentThread().isInterrupted*** and terminates if true

WebSocket

The application broadcasts live logs and updates using WebSocket. Use wss:// with SSL certificate

ws://host:port/queue/private

First, obtain JWT token from auth endpoint, then use the **token as value for Sec-WebSocket-Protocol** header when establishing the connection.

The app broadcasts two types of objects: LOG and UPDATE. The payload of LOG is always a string:

```
{
 "type": "LOG",
 "payload": "Script 1, Execution status: CANCELLED",
 "time": "2025-11-11T18:15:23.8588285"
 }
```

The payload of UPDATE may contain two types of objects used to notify another system that new results are available via REST.

```
{
   "type": "UPDATE",
   "id": 1, // sequential number of update
   "payload": { }
}
```

When new result is available, the **payload** contains

```
{
 "scriptId": 1,
 "resultId": 42,
 "timestamp": "2025-11-11T18:15:23.8588285",
 "status": "SUCCESS",
 "delivered": false
 }
```

When script is updated in the database, the payload contains:

```
{
 "id": 1,
 "name": "My Script",
 "group": "default",
 "compiles": true,
 "runs": 10,
 "fails": 2,
 "nextRun": "2025-11-12T08:00:00",
 "isEnabled": true,
 "isEnqueued": false
 }
```