EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

The Switchboard

Technical overview

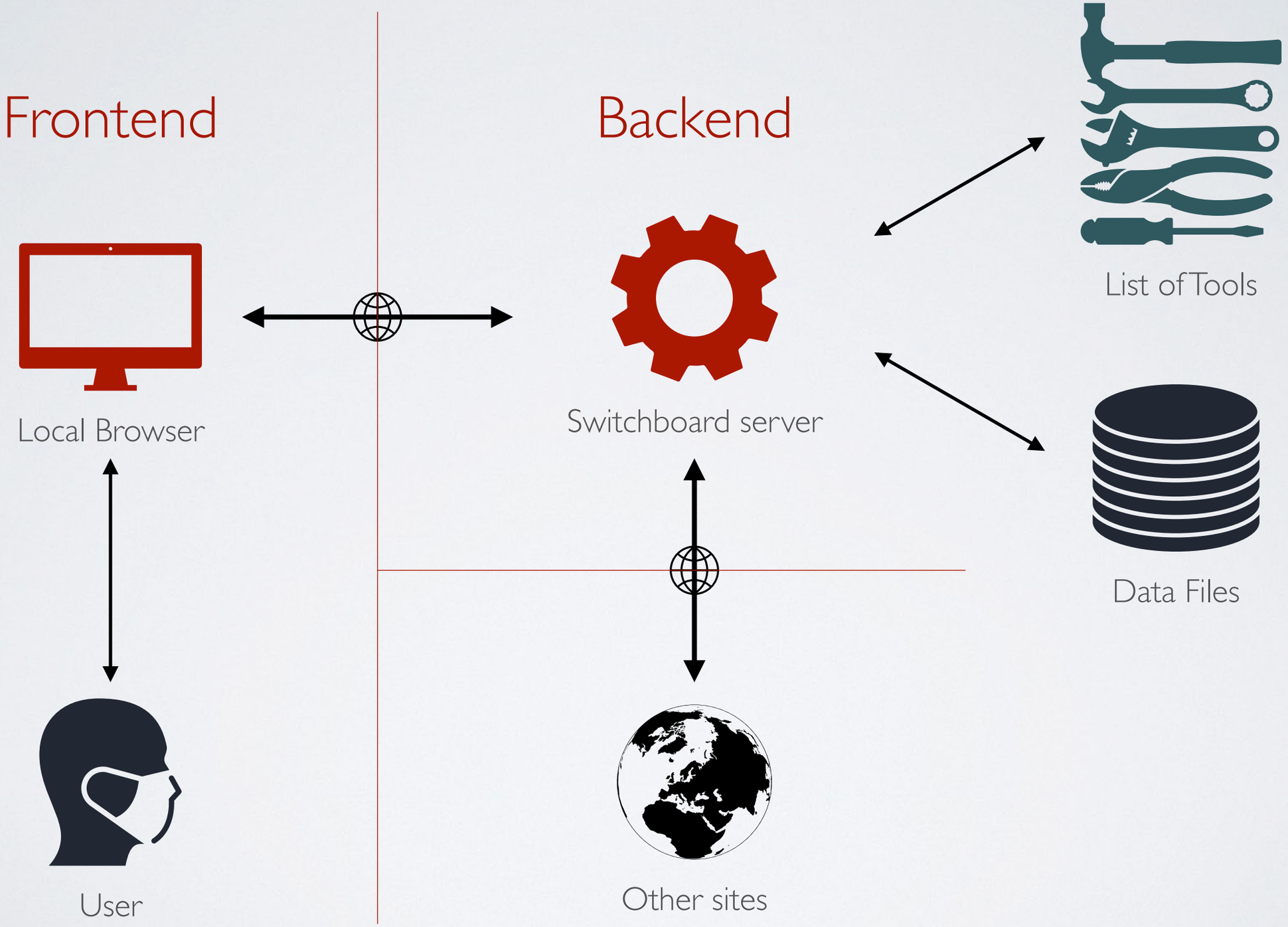
January 2022

please interrupt at any time!

User view

- Switchboard working modes
  - *classic view*, site rendered normally
    - responsive design: desktop and mobile layout
    - *popup mode*: rendered as mobile view into an iframe: VLO example
    - *dictionary mode*: popup mode combined with selection of special tools *(demo)*
- Special features:
  - archives extractions, decompression
  - text extraction

What's a Web App



Switchboard

- Repositories
  - Integration procedure
  - Popup/dictionary mode
  - Invoking Switchboard
- Tools
  - Tool description format
  - Update tools procedures
  - Automatic tools sync
  - Invoking the tools
- Instances
  - Production instance
  - Beta instance
  - SSHOC instance
- The Profiler
  - Apache Tika
  - Profile and features
- The WebApp
  - Frontend
    - Main view
    - Popup mode
    - Dictionary mode
    - Archives, text extraction
  - Backend
    - (Public) API
    - Matching algorithm
    - Data storage (CORS)
    - Configuration
- CI and DevOps
  - Docker
  - UI automatic tests

Code, Metadata, Docs

- Everything happens on GitHub:
  - Source code: github.com/clarin-eric/switchboard
  - List of Tools: github.com/clarin-eric/switchboard-tool-registry
  - Documentation: github.com/clarin-eric/switchboard-doc
  - Profiler project: github.com/weblicht/profiler

Repositories

- github.com/clarin-eric/switchboard-doc/blob/master/documentation/IntegrationProvider.md

- either use **integrated popup** [example]

- or **browser redirection** [example]

Instances

- **production** instance: switchboard.clarin.eu

- **beta** instance: beta-switchboard.clarin.eu

- **SSHOC**/development instance: weblicht.sfs.uni-tuebingen.de/switchboard-test

Tools

- github.com/clarin-eric/switchboard-tool-registry
  - tool descriptions (json files), logos (image files), schemas
  - *master* vs. *production* branch
    - update procedures: see the README
- Tool description spec, tool invocation API:
  github.com/clarin-eric/switchboard-doc/tree/master/documentation
- Automatic update of tools (from github to switchboard, using git)

Tools

- Human-oriented metadata


```
"id": 112,
"formatVersion": "2",
"task": "Text Analytics",
"deployment": "production",
"integrationType": "Integrated",
"name": "WebLicht Advanced Mode",
"logo": "weblicht.jpg",
"homepage": "https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Main_Page",
"location": "Tuebingen, Germany",
"creators": "CLARIN-D Centre at the University of Tuebingen, Germany",
"contact": {
    "person": "CLARIN WebLicht Support",
    "email": "wlsupport@sfs.uni-tuebingen.de"
},
"authentication": "Requires a CLARIN Service Provider Federation account, provided by many universities and
    institutions.",
"description": "This tool links to the WebLicht environment without preselecting an execution chain. WebLicht is an
    execution environment for automatic annotation of text corpora. Linguistic tools such as tokenizers, part of
    speech taggers, and parsers are encapsulated as web services, which can be combined by the user into custom
    processing chains. The resulting annotations can then be visualized in an appropriate way, such as in a table
    or tree format.",
```

Tools

- Machine-oriented metadata

```
"inputs": [
    {
        "id": "text",
        "mediatypes": [
            "text/plain",
            "text/rtf",
            "application/rtf",
            "application/msword",
            "application/vnd.openxmlformats-officedocument.wordprocessingml.document"
        ],
        "languages": "generic",
        "maxSize": 10240000
    }
],
"webApplication": {
    "url": "https://weblicht.sfs.uni-tuebingen.de/weblicht/",
    "queryParameters": [
        {
            "name": "input",
            "bind": "text/dataurl"
        },
        {
            "name": "lang",
            "bind": "text/language",
            "encoding": "639-1"
        }
    ]
}
```

Tools

- Local tools


```
"standaloneApplication": {
    "downloadURL": [
        {
            "url": "https://exmaralda.org/en/release-version/",
            "type": "release",
            "targetlang": "en"
        },
        {
            "url": "https://exmaralda.org/de/offizielle-version/",
            "type": "release",
            "targetlang": "de"
        },
        {
            "url": "https://exmaralda.org/en/release-version/",
            "type": "preview",
            "targetlang": "en"
        },
        {
            "url": "https://exmaralda.org/de/vorschau-version/",
            "type": "preview",
            "targetlang": "en"
        }
    ],
    "applicationSuite": "EXMARaLDA",
    "runtimeInformation": {
        "runtimeEnvironment": [
            "Java (included in newer versions)"
        ],
        "operatingSystem": [
```

Architecture

- **Front-end**: what is being run in the user's browser
  - Single page application, React with Redux
- **Back-end**: what runs on our internet server
  - Java app using Dropwizard (Jetty web server, Jersey REST framework, Jackson for JSON)
  - The **Profiler** library

Profiler

- Source: github.com/weblicht/profiler
  - Using: Apache Tika, Optimaize language detector
- Simple interface: given a file, returns its profile(s):

```
List<Profile> profile(File file) throws IOException, ProfilingException;
```

- What's a *profile*?
  - *MediaType*, e.g. "`text/plain`", "`application/pdf`"
  - *Language*: ISO 639-3 code, e.g. "`deu`"
  - Depending on format: ***version***, ***layers***, ***attributes***, etc.

Backend

- API and Public API:
  `/api/info, /api/tools[/:id], /api/tools/match`

- Data storage:
  - why? CORS issues
  - how? temporary copies of the data;
    with restrictions: known mediatype, max size, remove quickly

- Configuration options: see `config.yaml`

Backend: The matching algorithm

- Input:

  `list of profiles, one for each input data file`

- Output:

  ```
  list of tools that match and a list of mappings:
    a matching file for each tool's input
  ```

- Algorithm idea:

  ```
  for each file input profile, go through all tool input slots
  if the profile matches with the input slot:
    save the match as a partial result
    mark the matching profile and input slot as used
    recursively match the other file inputs and input slots
  ```

- Complications: `input slots matching multiple files; optional input slots`

Frontend

- *action* functions that populate the local Redux store with data from the server (sent as json)

- classic React ***pure*** components (data passed as arguments)

- wrapping containers (taking data from Redux store and passing it to components)

- a few special functions to build tool invocation URLs

- SCSS styling (easier to work with than css)

DevOps & CI

- use containerization (Docker) for building and deployment
  - advantage: reproducible builds, one package for all components, easy to deploy, easy to debug
  - disadvantage: learning another tool; future-unsafe?
- UI (end-to-end) tests
  - integrated in the github workflow, run on any branch push and pull request

Unsolved issues, future plans

- Data privacy cannot be guaranteed
  - Solution: Do not use private data?
- Obfuscated data access: landing pages, login required
  - Plan: use DOG for accessible data (DOIs, PIDs)
  - Possible solution: delegated access to restricted resources
- No matches found! 😡
  - Plan: Preflight API

Thanks! Q&A?