Devcontainer für dieses Quarto-Projekt

Um unabhängig von lokalen Installation zu sein, wurde eine devcontainer eingerichtet. Der Devcontainer wird als Docker Container gestartet. Das Projektverzeichnis wird in den Container ge-mounted. Beim Öffnen des Projektes mit VSCode, schlägt VSCode vor, den Devcontainer zu starten. Man arbeitet dann im lokalen Verzeichnis, hat aber über den Devcontainer die Tools für das Projekt zur Verfügung.

Alle notwendigen Dateien wurden unter intensiver Nutzung mit KI generiert, nicht alles ist vollständig reviewed und plausibilisiert. Bei den ersten Durchläufen, hat es aber funktioniert

Will man ohne Devcontainer arbeiten, so kann man das Projekt öffnen, ohne den Devcontainer zu starten. Dann nutzt man die selber lokal installierten Tools.

Kurzanleitung

Das ist nur eine kurze Anleitung. Die Details sind weiter unten erklärt.

Alle Befehle im Terminal von VSCode ausführen.

```
\CommentTok{\#Preview starten, um während der Entwicklung die Seite im Browser zu sehen}
\ExtensionTok{quarto}\NormalTok{ preview}

\CommentTok{\# Seite auf github publizieren, folgende Schritte sind notwendig:}
\CommentTok{\# {-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}}
\CommentTok{\# 1. Render nach docs}
\ExtensionTok{quarto}\NormalTok{ render}

\CommentTok{\# 2. weitere Dateien nach docs kopieren (Dateien zum Download)}
\ExtensionTok{python3}\NormalTok{ scripts/copy\_notebooks\_to\_docs.py }\AttributeTok{{-}{-}ext}\NormalTok{ ipynb,txt,pdf}

\CommentTok{\# 3. Links auf \textasciigrave{}.ipynb\textasciigrave{}{-}Dateien fixen (Quarto ersetzt diese durch html links)}
\CommentTok{\#Achtung im Zusammenhang mit preview funktioniert das nicht, da preview Seite erst rendert, wenn sie im Browser geöffnet wird.}
\ExtensionTok{python3}\NormalTok{ scripts/fix\_notebook\_links.py}

\CommentTok{\# Weitere Möglichkeiten...}

\CommentTok{\# Beispiele, wenn nur eine Datei ge{-}rendert werden soll}
\ExtensionTok{quarto}\NormalTok{ render index.qmd }\AttributeTok{{-}{-}execute} \AttributeTok{{-}{-}no{-}cache}
```

```
\ExtensionTok{quarto}\NormalTok{ render
files/lektionen\_hs25/251105\_symetricencryption/symetricencryption.qmd
}\AttributeTok{{-}{-}execute} \AttributeTok{{-}{-}no{-}cache}
\ExtensionTok{quarto}\NormalTok{ render files/lektionen\_hs25/251105\_symet
ricencryption/chapter/xor\_encryption.ipynb }\AttributeTok{{-}{-}execute}
\AttributeTok{{-}{-}no{-}cache}
```

Details zum Devcontainer

Diese DevContainer-Konfiguration stellt eine Entwicklungsumgebung bereit, in der Quarto und eine LaTeX-Umgebung (XeLaTeX) installiert sind. Damit kannst du die Website bauen und PDF-Ausgaben erzeugen. Weiter ist eine Phython Umgebung vorhanden, so dass die Jupiterlabs des Projektes ausgeführt werden können.

Dateien für den Devcontainer

- `.devcontainer/Dockerfile` — baut ein Ubuntu-basiertes Image mit Quarto CLI und einer minimalen TeX-Installation (`xelatex`).
- `.devcontainer/devcontainer.json` — VS Code DevContainer-Konfiguration.

Container starten

Schnellstart 1. Öffne das Projekt in VS Code. 2. Drücke F1 → `Remote-Containers: Reopen in Container` (oder verwende das grüne Remote-Icon). VS Code baut dann das Image (das erste Mal kann es einige Minuten dauern).

Quarto rendern

Quarto rendert die Daten aus dem Verzeichns _files: in das Verzeichnis *docs*. Pusht man das Projekt auf github, so wird das Verzeichnis docs via githubpages publiziert. https://skriptenmk.github.io/l_eW_24-28/

Auswahl von Befehlen für Quarto

- Terminal in VS-Code öffnen und Befehl eingeben

Um die Website lokal rendern und eine Vorschau zu starten:

```
\ExtensionTok{quarto}\NormalTok{ preview}
```

Um die komplette Seite zu rendern (statische Ausgaben in `docs/`):

```
\ExtensionTok{quarto}\NormalTok{ render}
```

Für PDF-Ausgabe einer einzelnen Datei

```
\ExtensionTok{quarto}\NormalTok{ render path/to/file.qmd
}\AttributeTok{{-}{-}to}\NormalTok{ pdf}
```

Extra: Notebooks/Assets in `docs/` kopieren

Wenn du sicherstellen willst, dass zusätzliche Dateien (z. B. `.ipynb`, `.txt`, `.pdf`) unter `docs/` landen und über GitHub Pages erreichbar sind, führe nach dem Rendern das mitgelieferte Kopier□Script aus.

Trockenlauf, ohne wirklich zu kopieren

```
\ExtensionTok{python3}\NormalTok{ scripts/copy\_notebooks\_to\_docs.py
}\AttributeTok{{-}{-}dry{-}run} \AttributeTok{{-}{-}ext}\NormalTok{
ipynb,txt,pdf}
```

Kopieren ausführen

```
\ExtensionTok{python3}\NormalTok{ scripts/copy\_notebooks\_to\_docs.py
}\AttributeTok{{-}{-}ext}\NormalTok{ ipynb,txt,pdf}
```

Zuerst Trockenlauf, dann kopieren

Du kannst beide Schritte auch in einer Zeile koppeln (die Kopie läuft nur, wenn das Rendern erfolgreich ist):

```
\ExtensionTok{quarto}\NormalTok{ render }\AttributeTok{{-}{-}execute}
\AttributeTok{{-}{-}no{-}cache} \KeywordTok{\&\&}
\ExtensionTok{python3}\NormalTok{ scripts/copy\_notebooks\_to\_docs.py
}\AttributeTok{{-}{-}ext}\NormalTok{ ipynb,txt,pdf}
```

Nachbearbeitung der Links

Das ein bisschen ein Hack (habe in der Kürze noch keine besser Lösung).

Manchmal wandelt Quarto in der gerenderten HTML Verweise auf Notebooks in `.html`-Links um. Aufgefallen ist mir das bei `.ipynb`-Dateien. Die Links haben aber die Idee die orginalen `.ipynb`-Dateien als Link zur Verfügung zu stellen. Dazu dient das folgende Script `copy_notebooks_to_docs.py`.

Das Skript sucht in `docs/` nach `.html`-Links und ersetzt sie durch relative Pfade zu vorhandenen `.ipynb`-Dateien im `docs/`-Baum.

Sicherheit: Das Skript ist vorsichtig und ersetzt standardmäßig keine Links, die auf tatsächlich vorhandene HTML-Seiten verweisen. Wenn du dennoch eine Ersetzung erzwingen willst (z. B. weil Quarto eine `.html`-Seite mit demselben Namen generiert hat, du aber stattdessen auf das `.ipynb` verlinken möchtest), verwende die Option `--force`.

Dry Run

```
\ExtensionTok{python3}\NormalTok{ scripts/fix\_notebook\_links.py
}\AttributeTok{{-}{-}dry{-}run}
```

Fix ausführen

```
\ExtensionTok{python3}\NormalTok{ scripts/fix\_notebook\_links.py}
```

force

```
\ExtensionTok{python3}\NormalTok{ scripts/fix\_notebook\_links.py
}\AttributeTok{{-}{-}force}
```

Hinweis

- Wenn du zusätzliche LaTeX-Pakete benötigst, kann es nötig sein, das Dockerfile zu erweitern (z. B. `texlive-pictures`, `texlive-lang-german` usw.).
- Die DevContainer-Konfiguration legt das Arbeitsverzeichnis unter `/workspace` und verwendet den Benutzer `vscode`.

Manueller Docker-Build (optional)

Wenn du VS Code nicht verwenden möchtest, kannst du das Image manuell bauen und einen Container starten:

```
\CommentTok{\# aus dem Ordner .devcontainer (ein Verzeichnis höher als das
Projekt{-}Root)}
\ExtensionTok{docker}\NormalTok{ build }\AttributeTok{{-}t}\NormalTok{
quarto{-}devcontainer }\AttributeTok{{-}f}\NormalTok{
.devcontainer/Dockerfile ..}

\ExtensionTok{docker}\NormalTok{ run }\AttributeTok{{-}{-}rm}
\AttributeTok{{-}it} \DataTypeTok{\textbackslash{}}
    \AttributeTok{{-}v} \StringTok{"}\VariableTok{$(}\BuiltInTok{pwd}\Variable
Tok{)}\StringTok{:/workspace"}
\DataTypeTok{\textbackslash{}}
```

```
    \AttributeTok{{-}w}\NormalTok{ /workspace
}\DataTypeTok{\textbackslash{}}
\NormalTok{    quarto{-}devcontainer bash}
```

Im Container kannst du dann `quarto render` oder `quarto preview` ausführen.