# Quasar Framework

# Agenda

- Quasar Einführung
- Quasar CLI
- Components
- Styling
- Layout
- Directives
- Plugins
- Hilfsfunktionen und Extensions

# Quasar Einführung

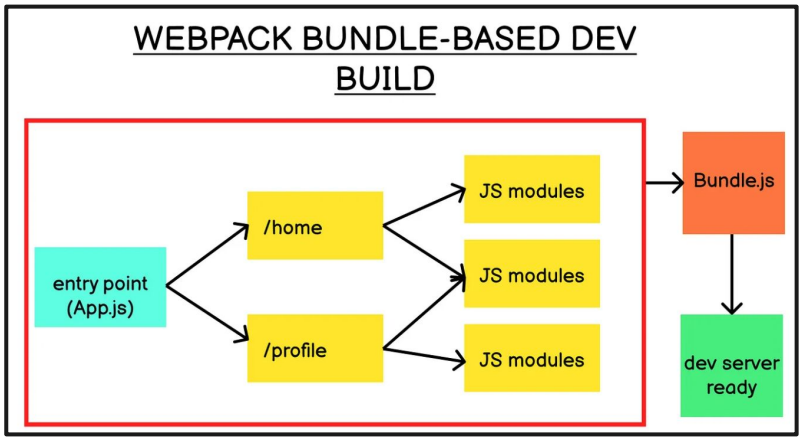
# Was ist Quasar?

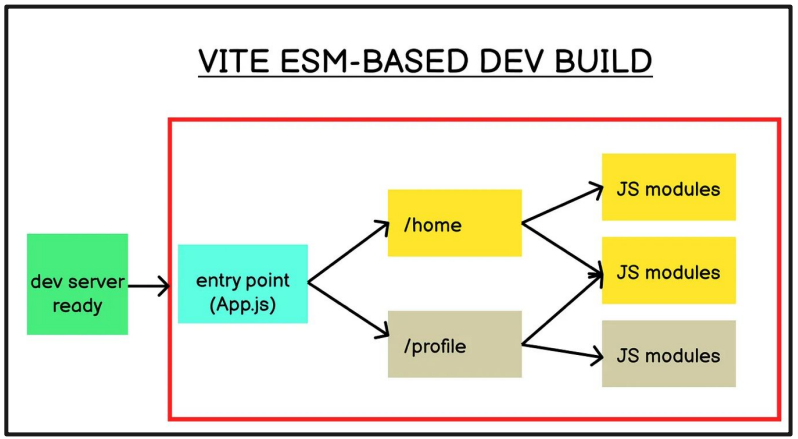
- JavaScript Framework, das auf VueJS basiert
- Eine Codebase, mehrere Plattformen
  - SPA (Single Page Application) → wird einmal geladen und dann nur dynamisch neu gerendert
  - SSR (Server-side rendered APP)
  - PWA (Progressive Web Application) → kann offline genutzt werden, Push-Benachrichtigungen, …
  - Browser Extension
  - Mobile Apps (Cordova or Capacitor)
  - Desktop Apps (Electron)
- Beinhaltet viele übliche Web Programming Methoden out of the box → Vermeidung von Boilerplate Code
- TypeScript Support

# Versionen

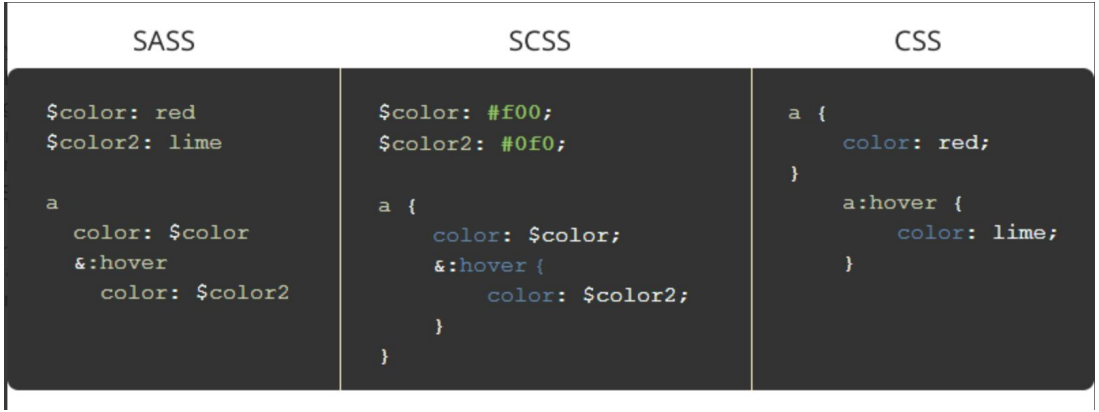
- Quasar CLI
  - Vollständiger Funktionsumfang
  - Einzige Möglichkeit für Multi Plattform mit einer Codebase
  - App Extensions verfügbar
- Quasar UMD
  - keine Installation notwendig
  - Beinhaltet nur Styling
- Vite Plugin
  - nicht eng in Quasar Komponenten integriert → keine native Unterstützung
- Vue CLI Plugin
  - nicht eng in Quasar Komponenten integriert → keine native Unterstützung

# Vite vs. Webpack



# Exkurs: SCSS/SASS




- Funktionalität ist die Gleiche
- SASS ist an YAML angelehnt
- SCSS entstand aus SASS und ist weiter verbreitet
- Syntax unterscheidet sich
- Variablen, Nesting und Schreiben von “Funktionen” möglich → wiederverwendbarer Code

# Quasar CLI

# Quasar Installation

VSCODE Plugins:
https://marketplace.visualstudio.com/items?itemName=vue.volar
https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint
https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode
https://marketplace.visualstudio.com/items?itemName=EditorConfig.EditorConfig

Voraussetzungen:

- NodeJS 14+
- Yarn V1: `npm install --global yarn`

Installation Quasar CLI:

```
yarn global add @quasar/cli
```

Create first project:

```
yarn create quasar
```

```
✔ What would you like to build? › App with Quasar CLI, let's go!
✔ Project folder: … QuasarTodo
✔ Pick Quasar version: › Quasar v2 (Vue 3 | latest and greatest)
✔ Pick script type: › Javascript
✔ Pick Quasar App CLI variant: › Quasar App CLI with Vite 6 (v2)
✔ Package name: … quasartodo
✔ Project product name: (must start with letter if building mobile apps) … Quasar
App
✔ Project description: … A Quasar Project
✔ Pick a Vue component style: › Composition API with <script setup>
✔ Pick your CSS preprocessor: › Sass with SCSS syntax
✔ Check the features needed for your project: › Linting (vite-plugin-checker + ESL
int), State Management (Pinia)
✔ Add Prettier for code formatting? … yes
✔ Install project dependencies? (recommended) › Yes, use yarn
```

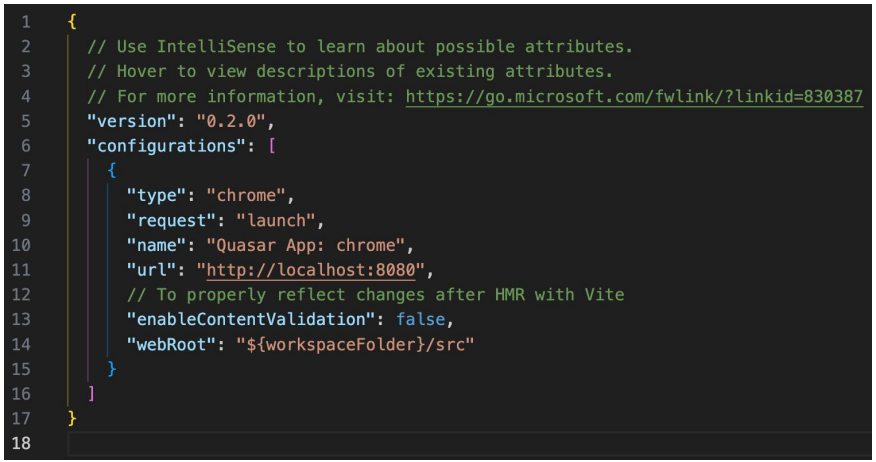
# Quasar CLI Commands

- quasar upgrade: Upgrade Quasar Projekt
- quasar info: Projektinformation über Versionen
- **quasar dev: Startet Development Server**
- **quasar build: App kompilieren und bauen**
- quasar clean: Alle gebauten Versionen aufräumen
- quasar new: Helfer für die Erstellung von pages, components und layouts
- quasar mode: Installieren oder Deinstallieren von Multiplattform Support
- quasar describe: Helfer für das Anzeigen der Quasar API
- quasar inspect: Debug Bundler Konfiguration, die von Quasar automatisch erstellt wird
- quasar ext: Installieren von App Extensions
- quasar run: Ausführen von Kommandos für App Extensions
- quasar serve: Starten eines https Webservers

Chrome Extension:
https://chromewebstore.google.com/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd

# Debugger Setup

- Klicke auf den Play-Button mit dem Käfer in der Menüleiste auf der linken Seite von VSCode
- Klicke auf: “Create a launch.json file”
- Wähle irgendeine Option aus
- Lege folgende Konfiguration an:



```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "chrome",
      "request": "launch",
      "name": "Quasar App: chrome",
      "url": "http://localhost:8080",
      // To properly reflect changes after HMR with Vite
      "enableContentValidation": false,
      "webRoot": "${workspaceFolder}/src"
    }
  ]
}
```

Test Debugger:

- Führe “quasar dev” im Projektordner aus
- Starte den Debugger mit dem grünen Play Btn
- Setze Breakpoint in MainLayout.vue
- Lade die Website neu

# quasar.config.js

- Zentrale Konfigurationsdatei von Quasar und allen Tools, die von Quasar verwendet werden
  - Entwicklungsserver
  - Quasar Komponenten, Directives, Plugins, …
  - Bundler Konfiguration (Vite)
  - Konfiguration der Multi Plattform Tools (Electron, Cordova, …)

# Ordnerstruktur

- public → statische Assets
- src → Quellcode der Website
  - assets → Dynamische Assets
  - components → Vue Komponenten, die in pages und layouts verwendet werden
  - css → CSS Dateien für die Website
  - layouts → Page- / Komponentenlayouts
  - pages → Seiten der Website
  - boot → boot Dateien
  - router → Vue Router
  - stores → Pinia Store
  - App.vue → Root Komponente für Vue
- index.html → Template für index.html (HTML Einstiegspunkt)
- dist → kompilierte Website

# Assets

/public vs /src/assets

- Assets in /src/assets werden nur kompiliert, wenn sie von einer .vue Datei referenziert werden (<img src="~assets/logo.png">)
- Assets in /public werden immer 1:1 kopiert (<img src="/logo.png">)

# Boot Files

```
export default ({ app, router, store }) => {
  // something to do
}
```

- Oft ist es nötig Code auszuführen, bevor die Root Vue App Instanz gestartet wird
- Use Cases: Initialisierung von Authentifizierung, Konfiguration und Initialisierung von Vue Libraries, etc…
- In Vue Projekten kann dieser Code üblicherweise in der Datei main.js ausgeführt werden
    - Quasar versteckt diese Datei, damit die Erstellung von Multi Plattform App möglich ist
- Boot Files können in quasar.config.js registriert werden
- /src/boot beinhaltet Boot Files

# Quasar Start Flow

1. Quasar wird initialisiert (components, directives, plugins, Quasar i18n, Quasar icon sets)
2. Quasar Extras werden importiert (Roboto font – if used, icons, animations, …)
3. Quasar CSS & globales CSS der App werden importiert
4. App.vue wird geladen (noch nicht benutzt)
5. Store: wird importiert
6. Store: wird in die Vue App Instanz eingefügt
7. Router: wird importiert
8. Boot files: werden importiert
9. Router: default export function wird ausgeführt
10. Boot files: default export function wird ausgeführt
11. Vue wird mit Root Komponente instantiiert und mit dem DOM verknüpft

# Umgebungsvariablen

- können mit “process.env” im Projektcode genutzt werden
- Werden in quasar.config.js definiert
- Wenn das Projekt mit “quasar build” kompiliert wird, werden if-Bedingungen für Umgebungsvariablen geprüft und aufgelöst

# Components

# Quasar Components

- Dokumentation: https://quasar.dev/components
- Gängige Komponenten, die in Websites gebraucht werden (Buttons, Tabellen, Karten, Formularelemente, …)
- Material Design Style
- Beinhalten Möglichkeiten den Inhalt, Style, Verhalten und Events von Komponenten zugänglich bzw. modifizierbar machen
    - **v-model:** 2-Wege-Bindung (wenn sich der Wert in der Eltern-Komponente ändert wird er automatisch in die Kind-Komponente übergeben und andersrum); müsste normalerweise mit Props / Events manuell implementiert werden
    - **Props:** Vue Component Props → ermöglichen einfache und dynamische Konfiguration der Komponenten
    - **Slots:** Vue Component Slots → Text oder Icons, die automatisch innerhalb der Komponente formatiert und platziert werden
    - **Events:** Vue Events → ermöglicht Ausführen von Funktionen, wenn ein Event der Komponente ausgelöst wird
    - **Methods:** Methoden, die direkt auf die Komponente ausgeführt werden können
    - **ComputedProps:** Properties der Komponente, die aktualisiert werden, wenn sich eine abhängige Property ändert

# Beispiel QInput Props

- Vue Component Property
- Platziert ein sichtbares Label am Textfeld
- v-model: Vue Technologie (2 Wege Binding)

```
<template>
  <q-page>
    <div class="q-pa-md" style="max-width: 400px">
      <q-input v-model="name" label="Your name *" />
    </div>
  </q-page>
</template>

<script setup>
import { ref } from "vue";
const name = ref("");
</script>
```

# Beispiel QInput Events

- Auf Events können Funktionen ausgeführt werden
- Das update:model-value Event wird immer ausgelöst, wenn eine Eingabe im Textfeld erfolgt

```
<template>
  <q-page>
    <div class="q-pa-md" style="max-width: 400px">
      <q-input
        v-model="name"
        @update:model-value="console.log('I am typing...')"
      />
    </div>
  </q-page>
</template>

<script setup>
import { ref } from "vue";
const name = ref("");
</script>
```

# Beispiel QInput Methoden

- Directive “ref” im Template⟶ Referenz auf die Komponente
- JavaScript Code: Variable mit gleichem Namen ⟶ Vue Instanz
- Quasar Methoden ⟶ Komponenten manipulieren

```vue
<template>
  <q-page>
    <div class="q-pa-md" style="max-width: 400px">
      <q-input ref="inputRef" v-model="name" />
      <q-btn
        class="q-mt-sm"
        label="Reset Validation"
        @click="reset"
        color="primary"
      />
    </div>
  </q-page>
</template>

<script setup>
import { ref } from "vue";
const name = ref("");
const inputRef = ref(null);

function reset() {
  inputRef.value.resetValidation();
}
</script>
```

# Beispiel QInput Slots

- Platzhalter für maßgeschneiderten Inhalt
- Slots: im Inneren der Komponente anlegen
- → <template v-slot:<slot-name>>

```
<template>
  <q-page>
    <div class="q-pa-md" style="max-width: 400px">
      <q-input v-model="name">
        <template v-slot:prepend>
          <q-icon name="font_download" />
        </template>
      </q-input>
    </div>
  </q-page>
</template>

<script setup>
import { ref } from "vue";
const name = ref("");
</script>
```

# Beispiel QInput ComputedProps

- Ähnlich wie Quasar Methoden
- Über ref der Vue Instanz im JavaScript nutzbar
- Wert, der in Abhängigkeit anderer Werte aktualisiert wird

```
<template>
  <q-page>
    <div class="q-pa-md" style="max-width: 400px">
      <q-input
        ref="inputRef"
        v-model="name"
        :rules="[
          (val) =>
            (val && val.length > 0 && !val.match(/^\d/)) ||
            'Please type something valid',
        ]"
        @blur="loseFocus()"
      />
    </div>
  </q-page>
</template>

<script setup>
import { ref } from "vue";
const name = ref("");
const inputRef = ref(null);

function loseFocus() {
  console.log(inputRef.value.hasError);
}
</script>
```

# Styling

# Themes & Theme Builder

- Theme is die Farbpalette einer Website
- Theme Builder ist ein Tool zum Erstellen eines Themes mit Hilfe von Visualisierung und Farbauswahl Tools
- https://quasar.dev/style/theme-builder

```
// quasar.config file

return {
  framework: {
    config: {
      brand: {
        primary: '#1976d2',
        secondary: '#26A69A',
        accent: '#9C27B0',

        dark: '#1d1d1d',
        'dark-page': '#121212',

        positive: '#21BA45',
        negative: '#C10015',
        info: '#31CCEC',
        warning: '#F2C037'
      }
    }
  }
}
```

# Typographie

- Zusätzliche Schriftarten können importiert werden
- CSS Helper Klassen zur Formatierung von Gewicht, Größe und Positionierung von Überschriften und Text

# Abstände

- CSS Helper Klassen für Abstände von Elementen
- Es besteht die Möglichkeit diese Klassen “breakpoint-aware” zu machen (Addon)

```
q-[p|m][t|r|b|l|a|x|y]-[none|auto|xs|sm|md|lg|xl]
    T       D                    S
```

# Farben

- Vorgefertigte Liste von Farben, die als SASS/SCSS Variablen genutzt werden können
-

```
<!-- Notice lang="sass" -->
<style lang="sass">
div
  color: $red-1
  background-color: $grey-5
</style>
```

# Weitere Styling Optionen

- CSS Klassen für Positionierung
- CSS Klassen für Sichtbarkeit
- Breakpoints für CSS für verschieden große Bildschirme
- CSS Klassen für Schatten

# Layout

# Flex Grid

```
<div class="row justify-center">
  <div class="col-12 col-md-2">
    .col-12 .col-md-2
  </div>
  <div class="col-12 col-md-auto">
    .col-12 .col-md-auto (Variable width content)
  </div>
  <div class="col-12 col-md-2">
    .col-12 .col-md-2
  </div>
</div>
```

- CSS Klassen für Zeilen und Spalten
  - Spaltensystem Größe 1-12 → mehr als 12 Spalten in einer Reihe → neue Zeile
  - Spaltenbreite konfigurierbar
  - Spaltenbreite auch variabel je nach Breakpoint konfigurierbar
  - Horizontale und vertikale Anordnung der Spalten in der Zeile konfigurierbar
- CSS Klassen für gleichmäßige Abstände zwischen Komponenten (Gutter)

# Layouts & Layout Builder

- Die QLayout Komponente bietet Konfigurationsoptionen für die komplette Website
  - Menu Bar
  - Side Drawers
  - Footer
- Layout Komponenten können verschieden angeordnet und/oder fixiert werden

# Directives

# Was ist ein Directive?

- Vue Funktionalität
- Ein Vue Directive wird als Objekt mit Lifecycle Hooks definiert
- Benutzerdefiniertes Attribut für HTML Elemente
- Kann auf jedes Element ausgeführt werden
- Quasar Beispiele: Ripple, Close Popup

# Plugins

# Was sind Quasar Plugins

- Können in der quasar.config.js geladen werden
- Können in der quasar.config.js konfiguriert werden
- Werden in der Regel in das Quasar App Objekt $q injected
- Bereitgestellte Methoden können über das Quasar App Objekt genutzt werden

# Quasar Plugins

- Meta: SEO Optimierung
- Notify: Darstellen von Snackbars verschiedener Art
- Loading und LoadingBar: Darstellung von Ladeprozessen
- Dialog: Erstellung von PopUp Dialogen
- Cookies: Cookie Handling
- ...

# Hilfsfunktionen & Extensions

# Icon Genie CLI

- Icon Genie ist ein Kommandozeilen Tool
- Nimmt ein Icon als Eingabeparameter
- Erstellt daraus skalierte Version in verschiedenen Größen und legt diese im Quasar Projekt ab

# Icon Libraries

- Kann via quasar.config.js out of the box installiert werden
- Support für mehrere Icon Bibliotheken, u.a.:
  - Material Icons
  - Material Symbols
  - Font Awesome
  - Ionicons
  - MDI
  - …

# Platform Identifizierung

$q.platform

- OS
- Mobile / Desktop
- Browser
- Versions

# Utilities

- Können als Funktion direkt von Quasar importiert werden
  - Datum
  - Farbe
  - DOM
  - Typ Validierung

# Assignment

Starte ein neues Quasar Projekt mit der Quasar CLI mit dem Titel “Movie Blog” und designe eine Landing Page für einen Movie Blog. Die Seite sollte eine Top Bar, ein Suchfeld und einen Footer im Layout enthalten. Der Seiteninhalt soll verschiedene Filme mit groben Informationen und einem entsprechenden Titelbild in einer oder mehreren Listen enthalten. Desweiteren soll das Durchschnittsrating des jeweiligen Films dargestellt werden. Beispiele für Listen sind z.B. “Top-Rangliste” oder “Trending Movies”. Die Daten zu diesen Filmen sollen aus der API kommen, die im Laufe der Vorlesung programmiert wird (zusammen mit Herrn Joschko).