Static Content

> **Verfügbarkeit**
> Ab Batix Application Framework Version 2.7.1 verfügbar.

Plugins können in ihre .zip Datei statische Ressourcen-Dateien wie HTML-Seiten, Bilder, Schriften oder JS-/CSS-Dateien integrieren. Diese müssen in einem Unterordner unterhalb des Projektordners namens `static` liegen. Falls im Plugin-Projektverzeichnis ein Ordner `static` existiert, wird dessen Inhalt automatisch in die ZIP übernommen.

Es können auch komplette Vue Apps oder auch mit VitePress erstellte Dokumentationen ausgeliefert werden. Dank Gradle landet sogar automatisch zur Build-Zeit die kompilierte Vue-App oder die erstellte VitePress-Seite in der .zip Datei. Siehe dazu auch den Guide Vue App ausliefern.

> **INFO**
> Hier ist die Rede von `RequestHandler`n, obwohl es im vorherigen Kapitel immer `RequestInterceptor` hieß. Das hat historische Gründe - `RequestHandler` war der alte Name und zugunsten einfacherer Backwards-Compatibility wurde hier der Name nicht angepasst - es handelt sich aber intern trotzdem um `RequestInterceptor`s.

```
fun registerRequestHandlerForStaticContent(
  pathPrefix: String,
  staticPath: String,
  fallback: String,
  guard: StaticContentHandler.Guard = null
)

fun registerRequestHandlerForStaticContent(
  condition: RequestCondition,
  pathMapper: Function<HttpServletRequest, String>,
  staticPath: String,
  fallback: String,
  guard: StaticContentHandler.Guard = null
)
```

Die erste Methode eignet sich für einfache Fälle. `pathPrefix` ist der Anfang des Pfads, auf den reagiert werden soll - dies ist kein regulärer Ausdruck, sondern ein normaler String. Es wird dabei

auf jede Domain reagiert. `staticPath` ist der Name des Ordners im `static` Ordner des Plugin-ZIPs. `fallback` ist ein optionaler Parameter, der eine Alternativ-Datei angibt, welche ausgeliefert wird, falls eine nicht-existierende Datei angefordert wird - bei SPAs ist dies meistens index.html. `guard` ist ein Callback, der das Functional Interface `StaticContentHandler.Guard` und damit die Methode `intercept` implementiert. Hier kann entschieden werden, ob die Ressource doch nicht (z. B. wegen fehlender Authorisierung) ausgeliefert werden soll (in diesem Fall sollte der Guard `true` zurückgeben).

Hat man als `pathPrefix` beispielsweise /import-plugin/frontend/ und als `staticPath` den Ordner "import-frontend" definiert (und `fallback` mit `null` angegeben), dann würde ein Aufruf von domain.tld/import-plugin/frontend/import.html die Datei <ZIP>/static/import-frontend/import.html ausliefern, falls diese existiert. Ein Aufruf von domain.tld/import-plugin/frontend/ (entspricht also dem `pathprefix`) liefert die index.html Datei (<ZIP>/static/import-frontend/index.html) aus.

Mit der zweiten Methode hat man dank einer `RequestCondition` mehr Kontrolle, unter welchen Pfaden die statischen Dateien ausgeliefert werden. Dafür muss man mittels `pathMapper` aber auch eine Funktion übergeben, die aus dem Request den entsprechenden Pfad zur statischen Datei extrahiert (da es ja kein festes Prefix gibt).

Wird eine angefragte Datei nicht gefunden und es ist kein `fallback` angegeben (oder `fallback` wird auch nicht gefunden), wird eine Fehlerseite mit Status 404 erzeugt. Dateien werden mit passendem MIME-Type und Cache-Headern ausgeliefert.

Der Aufruf ohne Dateiangabe (also z. B. domain.tld/import-plugin/frontend/) liefert die Datei index.html aus (falls diese existiert).

Ein in beiden Fällen optionaler `guard` kann die `intercept`-Methode überschreiben und auf Request, Response sowie den Ressourcennamen reagieren um die Auslieferung ggf. zu unterbinden (`return true` heißt blockieren).

```
fun interface Guard {
  fun intercept(
    request: HttpServletRequest,
    response: HttpServletResponse,
    resource: String
  ): Boolean
}
```

Beispiel

Es wird folgender Inhalt der Plugin-ZIP angenommen (Ausschnitt).

```
<ZIP>
└── static
    └── import-frontend
        ├── css
        │   └── bootstrap.min.css
        ├── import.html
        ├── index.html
        └── js
            └── bootstrap.min.js
```

Der folgende Handler wird registriert.

```
override fun load() {
  registerRequestHandlerForStaticContent(
    "/import-plugin/frontend/", // pathPrefix
    "import-frontend", // staticPath
    "index.html" // fallback
  )
}
```

Daraus ergeben sich folgende Request-zu-Datei Mappings.

Request	Datei
domain.tld/import-plugin/frontend/import.html	<ZIP>/static/import-frontend/import.html
domain.tld/import-plugin/frontend/	<ZIP>/static/import-frontend/index.html
domain.tld/import-plugin/frontend/existiert.nicht	<ZIP>/static/import-frontend/index.html

Innerhalb der .html Dateien können die Scripts und Styles mittels relativem Pfad angesprochen werden.

```
<link rel="stylesheet" href="css/bootstrap.min.css" />
<script src="js/bootstrap.min.js"></script>
```