Vue App

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

Vue.js Anwendungen bestehen aus JavaScript, HTML und CSS. Singe File Components bieten sogar die Möglichkeit diese drei Sachen für eine Komponente in einer einzigen .vue Datei zu definieren. Das hilft Ordnung zu schaffen, benötigt aber einen Build-Step, d. h. es muss aus dieser .vue Datei erst etwas generiert werden, womit der Browser etwas anfangen kann.

Es ist zwar auch möglich Vue-Templates (der HTML-Teil einer Komponente oder App mit Platzhaltern) zur Laufzeit im Browser compilen zu lassen, dies geht aber auf die Performance und bläht das finale JavaScript auf, da der Compiler mitgeliefert werden muss. Besser ist es also, schon zur Build-Zeit diese aufwendigen Schritte zu durchlaufen. Außerdem stehen so noch weitere Features zur Verfügung, wie z. B. npm-Dependencies oder CSS-Transformatoren.

Die Seite Static Content beschreibt die Möglichkeit mittels Plugins, fertig generierte, statische Dateien unter einem bestimmten Pfad auszuliefern. Wie diese Dateien zur Build-Zeit erzeugt werden können, beschreibt dieser Guide.

Vorbereitungen

Auf dem Entwicklungs-Computer wird eine aktuelle Node.js Version benötigt, um eine neue Vue App anzulegen. Die eingesetzte IDE (z. B. VS Code oder IntelliJ IDEA) sollte auch aktuell sein.

Außerdem muss schon das Grundgerüst eines Plugins vorhanden sein (siehe IDE Setup und Plugin). Ein Ordner names "static" muss nicht angelegt werden, da Gradle so konfiguriert wird, dass es beim Anlegen der .zip Datei die Dateien selbst vom richtigen Ort holt, nachdem diese dort erstellt wurden.

Vue App erzeugen

Das Erzeugen der Vue-Anwendung ist nicht Teil dieses Guides. Es sei an dieser Stelle auf die offizielle Vue Doku verwiesen. Im weiteren Text wird davon ausgegangen, dass die Vue-App im Ordner `vue/import-frontend` (ausgehend vom Hauptprojekt) erzeugt wurde. Es wird hier keine `build.gradle.kts` Datei erzeugt, da die Vue-App nicht als separates Gradle-Unterprojekt angelegt wird, sondern die Build-Steps direkt als Tasks im entsprechenden Plugin angelegt werden.

Build Tasks

In der `build.gradle.kts` Datei des Plugins, welches die Vue-Anwendung ausliefern soll, wird folgender Block ergänzt:

```
//
// -- node --
//

val npmInstall by tasks.registering(Exec::class) {
  val nodeProjectDir = "$rootDir/vue/import-frontend"
  workingDir = file(nodeProjectDir)

  inputs.file("$nodeProjectDir/package.json")
  inputs.file("$nodeProjectDir/package-lock.json")

  outputs.dir("$nodeProjectDir/node_modules")

  val isWindows = org.apache.tools.ant.taskdefs.condition.Os.isFamily(
    org.apache.tools.ant.taskdefs.condition.Os.FAMILY_WINDOWS
  )
  commandLine(
    if (isWindows) "npm.cmd" else "npm",
    "install"
  )
}

val npmRunBuild by tasks.registering(Exec::class) {
  val nodeProjectDir = "$rootDir/vue/import-frontend"
  workingDir = file(nodeProjectDir)

  group = "build"
```

```
  dependsOn(npmInstall)

  inputs.dir("$nodeProjectDir/public")
  inputs.dir("$nodeProjectDir/src")
  inputs.file("$nodeProjectDir/package.json")
  inputs.file("$nodeProjectDir/package-lock.json")
  inputs.file("$nodeProjectDir/vite.config.js")

  outputs.dir("$nodeProjectDir/dist")

  val isWindows = org.apache.tools.ant.taskdefs.condition.Os.isFamily(
    org.apache.tools.ant.taskdefs.condition.Os.FAMILY_WINDOWS
  )
  commandLine(
    if (isWindows) "npm.cmd" else "npm",
    "run",
    "build"
  )

  doFirst {
    createVersionFile()
  }
}

fun createVersionFile() {
  val envVersionFile = Paths.get("$nodeProjectDir/env/.env.local.version")
  if (envVersionFile.parent != null) {
    Files.createDirectories(envVersionFile.parent)
  }
  val envVersionFileContent = "VERSION=${project.version}"
    .replace("dirty", "d") // "-dirty" → "-d" damit sich kein Frontend user wundert
  Files.write(envVersionFile, envVersionFileContent.toByteArray())
}
```

Die beiden hier definierten Tasks sind vom Typ `Exec`, führen im Hintergrund also einfach das entsprechende `npm` Command aus. Liegt die Vue-App in einem anderen Verzeichnis, müssen lediglich die zwei `nodeProjectDir` Zeilen (6 & 24) angepasst werden.

Besonderes Augenmerk muss auf die `inputs` Zeilen im zweiten Task (`npmRunBuild`) gelegt werden. Hier sind alle Dateien und Verzeichnisse aufzuführen, die dafür sorgen, dass sich die Vue-Anwendung ändert. Das sind also beispielsweise JS/Vue Quellcodes, CSS-Dateien, statische Assets,

aber auch Dateien, die zur Build-Konfiguration der Vue-App genutzt werden. Die Liste der Inputs darf ruhig länger sein, Hauptsache alle Dateien sind erfasst, damit Gradle die App auch bei Änderungen neu baut. Hier aufgeführte Dateien müssen existieren, sonst meldet Gradle einen Fehler.

Bei Quasar-Projekten muss das `outputs.dir` zu `"$nodeProjectDir/dist/spa"` geändert werden.

Längeres Beispiel für Inputs

```
    inputs.file("$nodeProjectDir/.env.development")
    inputs.file("$nodeProjectDir/.env.development.local")
    inputs.file("$nodeProjectDir/.eslintignore")
    inputs.file("$nodeProjectDir/.eslintrc.js")
    inputs.file("$nodeProjectDir/.postcssrc.js")
    inputs.file("$nodeProjectDir/babel.config.js")
    inputs.file("$nodeProjectDir/jsconfig.json")
    inputs.file("$nodeProjectDir/package.json")
    inputs.file("$nodeProjectDir/package-lock.json")
    inputs.file("$nodeProjectDir/quasar.conf.js")
    inputs.file("$nodeProjectDir/quasar.extensions.json")
```

Um die Version des Git-Tags im Vue-Projekt auslesen zu können, wird beim build eine `env/.env.local.version`-Datei im vue-Ordner erzeugt. Im Falle z.B. eines Quasar-Projektes muss in der `quasar.config.js` noch unter `build.envFiles` der Name der Datei (als String-Array) ergänzt werden, damit der Inhalt der Datei in `process.env` aufgenommen wird.

Der bereits angelegte Task `packageBatixPlugin` wird um den folgenden `from` Block ergänzt:

```
from(npmRunBuild) {
  into("static/import-mitarbeiter-frontend")
}
```

> Gradle Task Caching
> Gradle Tasks können definieren, was deren Input- und Output-Dateien sind. Anhand der Inputs kann Gradle bestimmen, ob ein Task überhaupt laufen muss, oder nicht. Wenn sich die Inputs seit dem letzten Run nicht geändert haben, ist der Task `UP-TO-DATE` und muss nicht noch einmal laufen. Bedingung ist natürlich, dass die Inputs korrekt erfasst wurden.
>
> Im ersten Task (`npmInstall`), der die benötigten NPM Packages installiert, wurde `package.json` als Input definiert. Sobald dort also eine neue Dependency eingetragen wurde,

> sieht Gradle, dass sich die Datei geändert hat, und führt den Task erneut aus.
>
> Dank der Outputs müssen andere Tasks nicht genau wissen, welche Dateien ein Task erzeugt, um diese beispielsweise in ein Archiv zu kopieren. Dieser Umstand wird hier im `packageBatixPlugin` Task genutzt.

Plugin Code

Damit die statischen Dateien auch ausgeliefert werden, ist die `load()` Methode in der Plugin-Hauptklasse um folgenden Code zu erweitern

```
registerRequestHandlerForStaticContent(
  "/mitarbeiter-plugin/import/",
  "import-mitarbeiter-frontend",
  null
)
```

Der erste Parameter (`pathPrefix`) muss dem Public Path der Vue-Anwendung entsprechen (bei Quasar: `quasar.config.js: build: publicPath`). Der zweite Parameter (`staticPath`) muss zur `into` Zeile aus dem `from` Block oben passen. Den dritten Parameter (`fallback`) lassen wir hier leer. Falls die Vue-App aber eine Single Page Application (SPA) ist, dann sollte hier `index.html` stehen.

Aufruf

Wird nun das Plugin über den Gradle Task `build` erstellt, ins System hochgeladen und aktiviert, erscheint die Vue-App unter dem angegebenen Pfad. □□