Bei Verwendung des MVC-Musters können z. B. zwei verschiedene Endgeräte das gleiche Model verwenden; der View wird z. B. einmal für die Desktop-Anwendung und einmal für das mobile Endgerät implementiert.

Bei Webanwendungen erstreckt sich das MVC-Muster über Server und Client und ist damit etwas komplexer als das klassische MVC-Muster, wie in Abbildung 5.13 zu erkennen. Der Browser stellt den View als grafisches Element dar, der Controller ist meistens JavaScript-Code, der die Benutzeraktionen verarbeitet und diese gegebenenfalls an das Model weitergibt. Weitere Informationen zum MVC-Muster finden Sie in Abschnitt 6.1.2, »Generierung des Model-View-Controller-Musters in SAPUI5«.

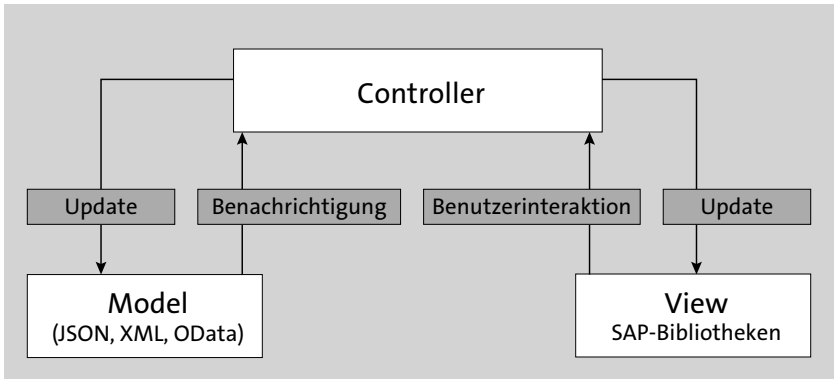

Abbildung 5.13 MVC-Architekturmuster in der Webentwicklung

5.3 SAPUI5-Entwicklungsprojekt anlegen

Das MVC-Muster ist ein wesentlicher Bestandteil der SAPUI5-Entwicklung, da SAPUI5-Anwendungen im MVC-Muster implementiert werden.

5.3.1 Projekt im SAP Business Application Studio anlegen

Die Entwicklung von SAPUI5-Anwendungen im SAP Business Application Studio wird in sogenannten Projekten organisiert. Wenn Sie ein Beispiel aus dem Buch nachprogrammieren möchten, dann legen Sie dafür am besten jeweils ein neues Projekt an. Später entspricht jedes Projekt einer SAPUI5-Anwendung oder Bibliothek.

Um im SAP Business Application Studio ein neues Projekt anzulegen, gehen Sie entweder über das Menü **File • New Project from Template** ❶, oder Sie klicken auf den Button **Start from Template** ❷ (siehe Abbildung 5.14).

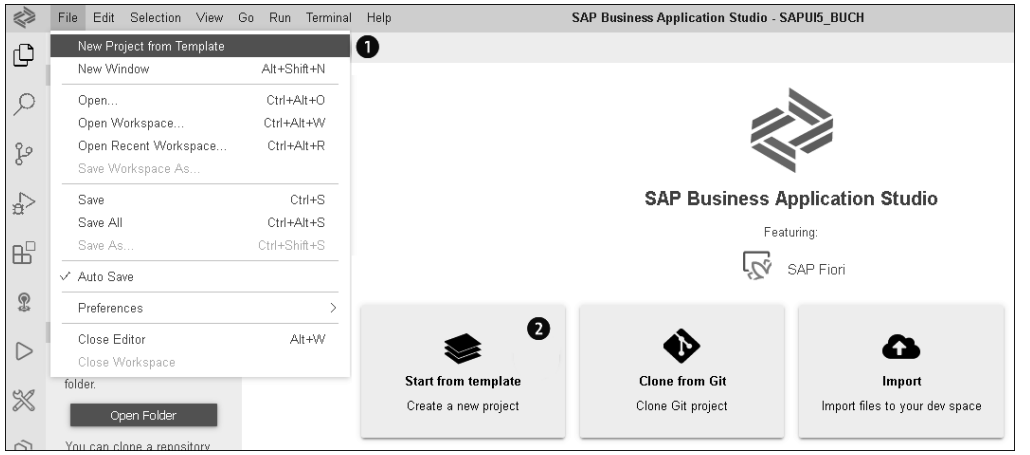

Abbildung 5.14 Neues Projekt im SAP Business Application Studio anlegen

Es öffnet sich ein Wizard, wählen Sie im ersten Schritt die Option **SAP Fiori application**, und klicken Sie auf **Start** (siehe Abbildung 5.15).

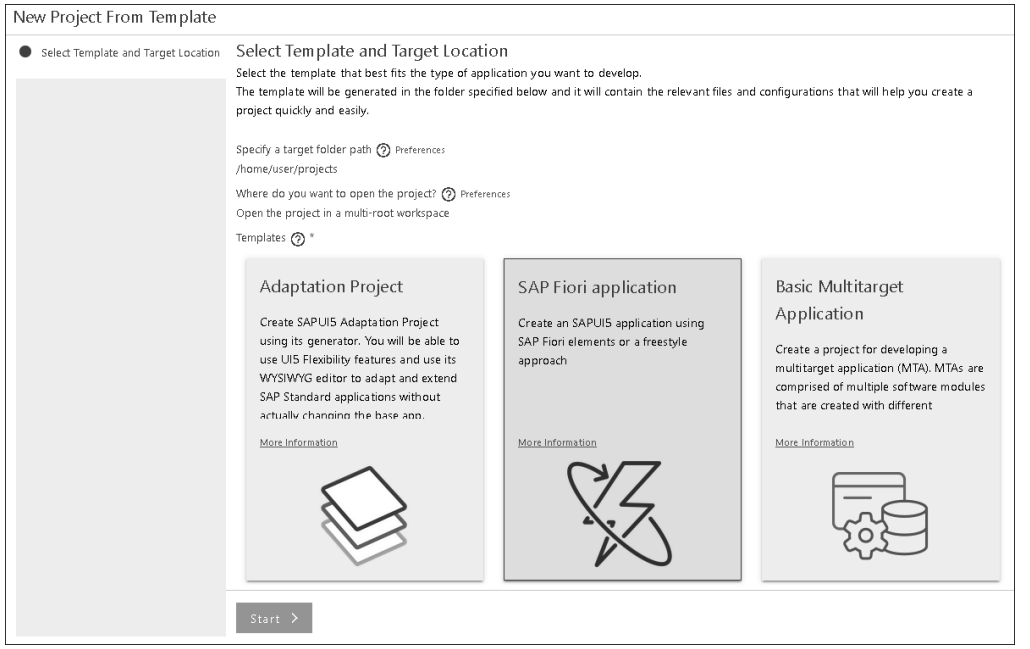

Abbildung 5.15 »SAP Fiori application« als Template auswählen

Im nächsten Schritt wählen Sie im Feld **Application Type** die Option **SAPUI5 freestyle** ❶, als Floorplan die Option **SAPUI5 Application** ❷ und klicken Sie auf **Next** ❸, um zum nächsten Bildschirm zu gelangen (siehe Abbildung 5.16).

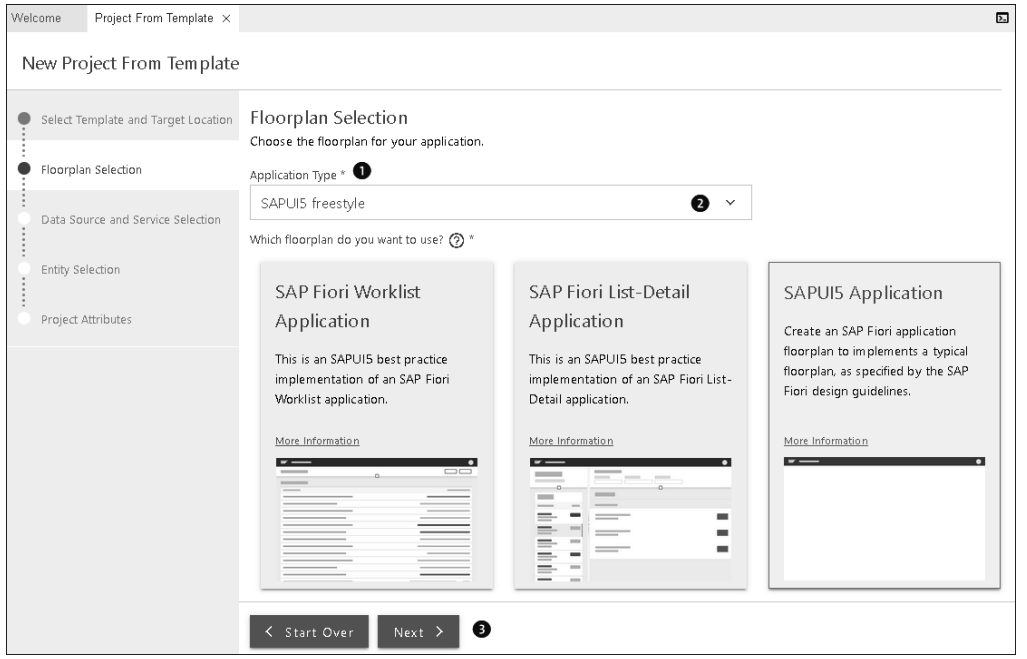

Abbildung 5.16 Freestyle-SAPUI5-Anwendung als Typ auswählen

Wenn wir in Kapitel 8 eine SAPUI5-Anwendung mit OData-Services entwickeln, werden wir in diesem Schritt die entsprechende Datenquelle auswählen. Für die ersten Gehversuche wählen Sie hier jedoch im Feld **Data source** die Option **None** und klicken auf **Next** (siehe Abbildung 5.17).

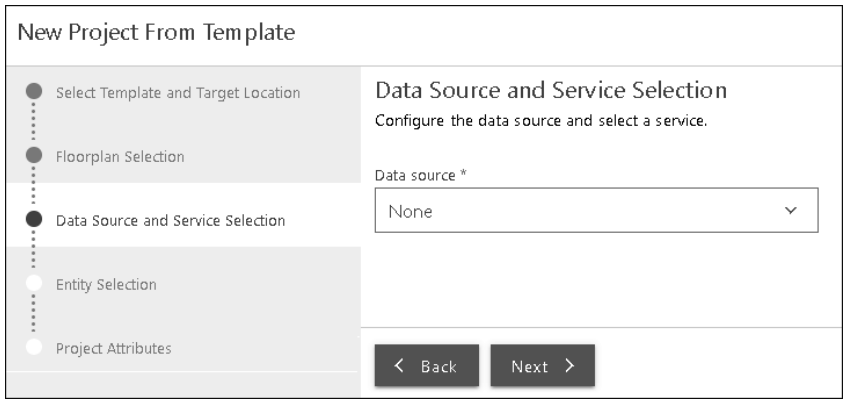

Abbildung 5.17 Ohne Datenquelle fortfahren

Im nächsten Schritt legen Sie den View-Namen an. Geben Sie dazu im Feld **View name** den Namen »Main« ein, und klicken Sie auf **Next** (siehe Abbildung 5.18).

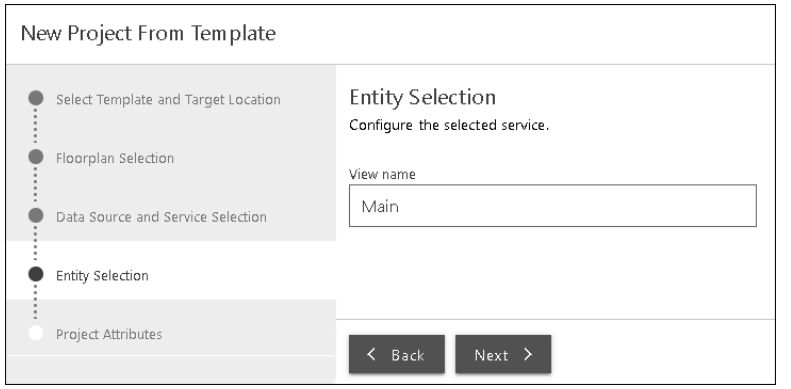

Abbildung 5.18 View-Namen festlegen

Im letzten Schritt müssen Sie noch den Namen der Anwendung festlegen. Ich habe für dieses erste Beispiel den Namen »hello.world« ausgewählt. Der Name sollte sprechend und selbsterklärend sein, sodass Sie das Projekt schnell wiederfinden. Alle anderen Einstellungen lassen Sie erst einmal bei den Standardeinstellungen, wir kommen zu gegebener Zeit darauf zurück. Legen Sie das Projekt mit **Finish** an (siehe Abbildung 5.19).

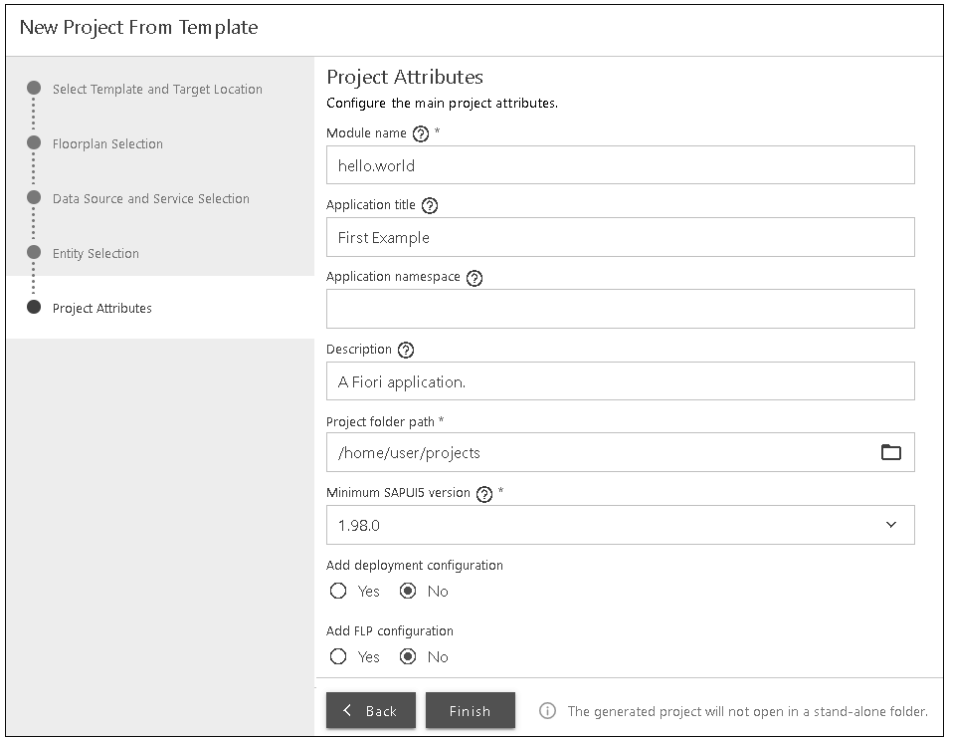

Abbildung 5.19 Namen der Anwendung festlegen

Den Namensraum der Anwendung (Feld **Application namespace**) benötigen Sie, wenn Sie eine SAP-Fiori-App erstellen möchten. Wenn Sie die Anwendung ins SAP Fiori Launchpad einhängen, wird diese über den eindeutigen Namespace identifiziert. Für den ersten Moment können wir das Feld aber leer lassen. Eine ausführliche Erläuterung zu diesem Vorgehen finden Sie in Kapitel 10, »Beispielentwicklung einer SAP-Fiori-App«.

Das Projekt wird nun angelegt. Wenn alles fertig ist, wird Ihnen eine Übersichtsseite, wie in Abbildung 5.20 dargestellt, angezeigt.

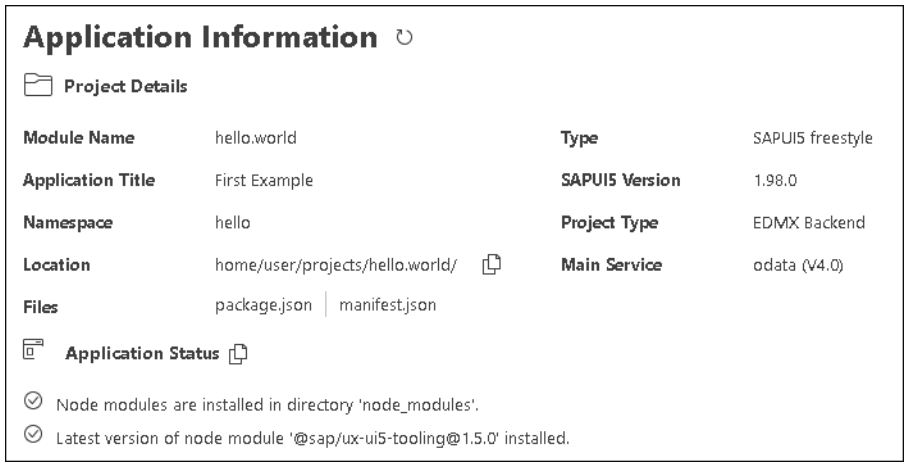

Abbildung 5.20 Angelegtes Projekt

5.3.2 Projekt öffnen

Um Ihr gerade angelegtes oder aber auch jedes andere Projekt zu öffnen, nutzen Sie das Menü **File • Open** ❶. Wenn Sie die Grundeinstellungen nicht verändert haben, finden Sie Ihre Projekte auch im Verzeichnis **projects** ❷. Markieren Sie das entsprechende Projekt, in unserem Beispiel **hello.world**, und klicken Sie auf **Open** ❸ (siehe Abbildung 5.21). Links öffnet sich nun das Projekt mit der entsprechenden Struktur, doch dazu in Abschnitt 5.3.5, »Projekt in Visual Studio Code anlegen«, mehr.

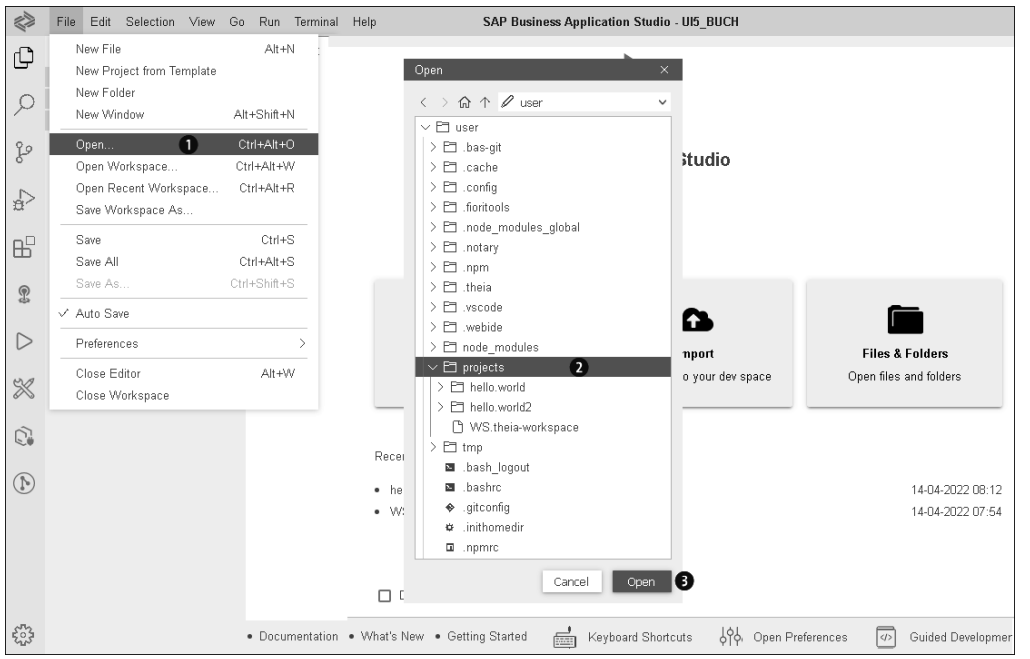

Abbildung 5.21 Projekt öffnen

5.3.3 Anwendung im SAP Business Application Studio ausführen

Wenn Sie Ihre Anwendung ausführen möchten, können Sie entweder eine entsprechende Konfiguration anlegen und ausführen, indem Sie auf den Ausführen-Button klicken ❶, oder Sie starten direkt per Kommandozeile, indem Sie das Menü öffnen und dort **Preview Application** ❷ auswählen (siehe Abbildung 5.22).

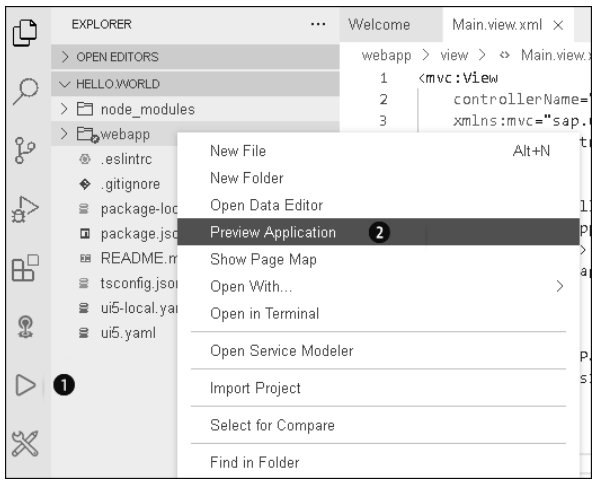

Abbildung 5.22 Anwendung ausführen

Um eine Konfiguration anzulegen, drücken Sie auf der rechten Seite auf den Menüeintrag **Run Configuration** und im Anschluss auf **Create Configuration**, wie in Abbildung 5.23 dargestellt.

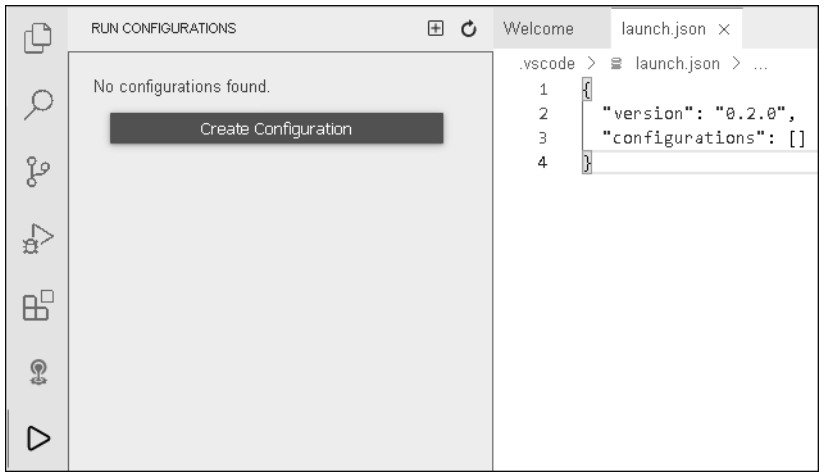

Abbildung 5.23 Run-Konfiguration anlegen

Wählen Sie im Folgebild (siehe Abbildung 5.24) Ihr Projekt (hier **hello.world**) aus, wählen Sie **index.html** als Dateiname aus, und speichern Sie diese Konfiguration.

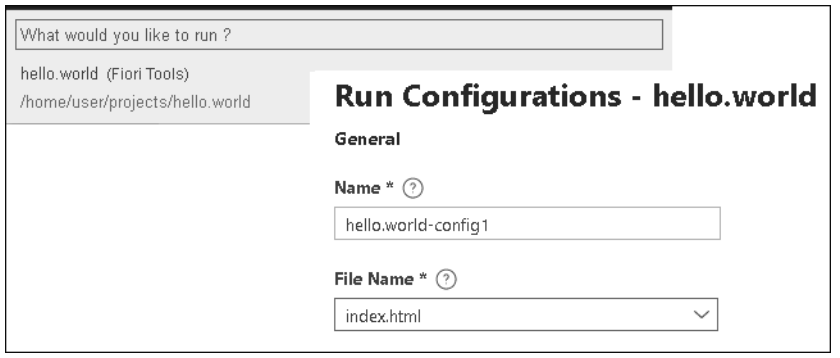

Abbildung 5.24 Run-Konfiguration anlegen – Wizard

Wenn Sie nun auf den Ausführen-Button klicken, werden Ihnen die angelegten Konfigurationen angeboten und Sie können durch einen Klick auf den Pfeil die entsprechende Konfiguration starten (siehe Abbildung 5.25).

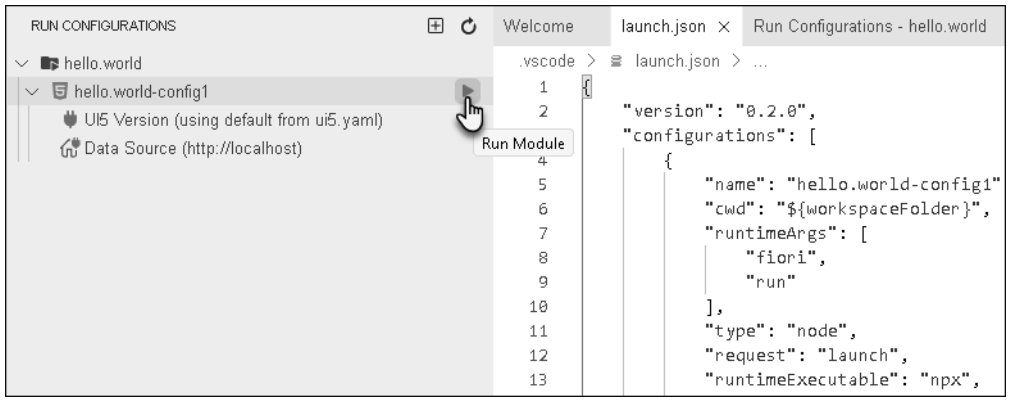

Abbildung 5.25 Konfigurierten Run starten

Vielleicht fragen Sie sich nun, was das Ganze soll. Wenn Sie sich mit SAPUI5 besser auskennen, werden Sie mit Sicherheit Unit-Tests, OPA5-Tests oder Ähnliches implementieren. Dann können Sie z. B. eine Konfiguration anlegen, die die entsprechenden Unit-Tests ausführt. Das UI5-Tooling bringt z. B. bereits fünf fertige Konfigurationen mit (siehe Abbildung 5.26).

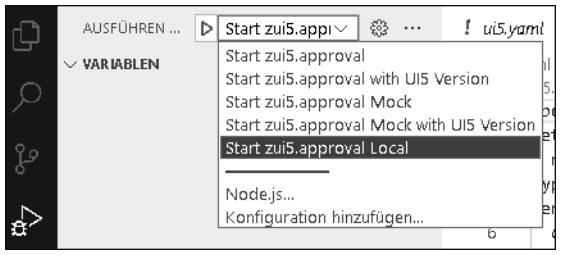

Abbildung 5.26 Vorkonfigurierte Run-Optionen

Der normale Start der App, der Start mit einer spezifischen SAPUI5-Version, diese Konfiguration erwartet die SAPUI5-Version als Argument, wie Ihnen ein Blick in die Konfiguration verrät (siehe Listing 5.1), und das Gleiche noch mal, allerdings mit Mockdaten.

```
        ],
        "args": [
            "--",
            "${input:UI5Version}"
        ],
```

Listing 5.1 Übergabe der SAPUI5-Version als Argument

5.3.4 Vorhandenes Projekt ins SAP Business Application Studio importieren

Wenn Sie die Lösung aus dem Buch importieren möchten, gehen Sie wie folgt vor. Gehen Sie auf die Seite *www.sap-press.de/5501*. Auf der Seite finden Sie den Abschnitt **Materialien zum Buch**, wo Sie die Beispiele als ZIP-Datei herunterladen können. Entpacken Sie die heruntergeladene ZIP-Datei, und navigieren Sie zu dem Verzeichnis **Listing_Kap4**, dort befindet sich eine gepackte Datei **hello.world.zip**. Diese Datei müssen Sie nicht entpacken, Sie können sie direkt in das SAP Business Application Studio importieren.

Wählen Sie dazu auf der Startseite des SAP Business Application Studios den Button **Import** aus, wie in Abbildung 5.27 dargestellt.

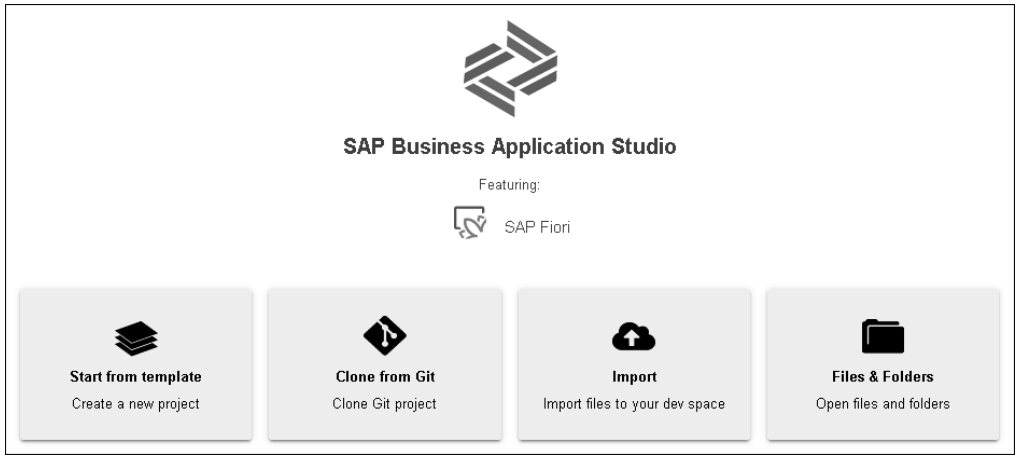

Abbildung 5.27 Beispielprojekt importieren

Sollte die entsprechende Willkommensseite nicht angezeigt werden, öffnen Sie diese über das Menü **Help • Welcome** (siehe Abbildung 5.28).

Abbildung 5.28 Welcome-Startseite öffnen

Wählen Sie die entsprechende ZIP-Datei aus, in unserem Beispiel **hello.world.zip**, diese wird anschließend in die Cloud hochgeladen. Nach dem erfolgreichen Upload erscheint ein kleines Fenster, wie in Abbildung 5.29 dargestellt. Bestätigen Sie dieses mit **Open in New Workspace**.

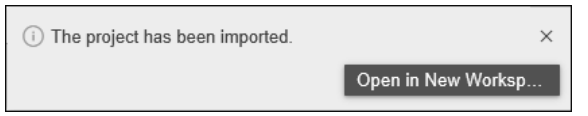

Abbildung 5.29 Importiertes Projekt öffnen

Damit Sie das Beispiel ausführen können, müssen Sie noch die *Node*-Pakete (npm) installieren. Was das genau ist, haben Sie bereits in Abschnitt 4.2, »Visual Studio Code«, erfahren.

Gehen Sie dazu mit der rechten Maustaste auf das Projekt, und wählen Sie **Open in Terminal** aus (siehe Abbildung 5.30).

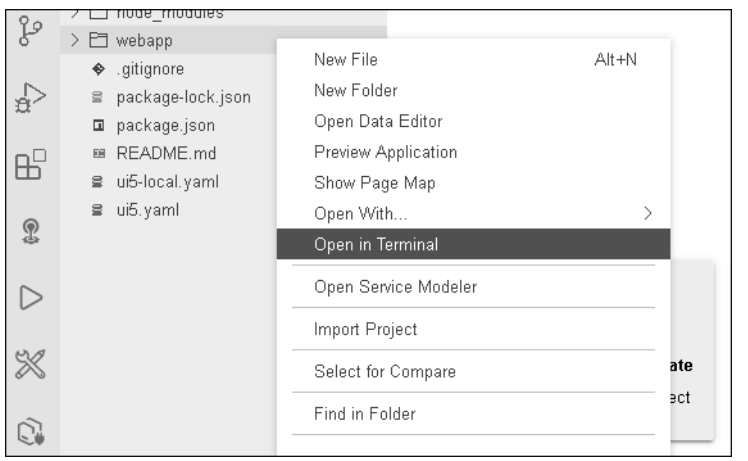

Abbildung 5.30 Terminal öffnen

Tippen Sie anschließend im Terminalfenster »npm install« ein, und drücken Sie die ↵-Taste, wie in Abbildung 5.31 dargestellt.

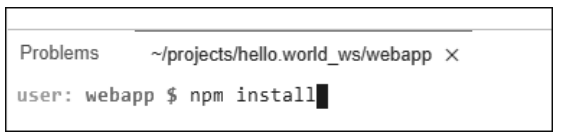

Abbildung 5.31 Pakete installieren

Im Anschluss können Sie die Anwendung, wie in Abschnitt 5.3.2, »Projekt öffnen«, dargestellt, starten.

5.3.5 Projekt in Visual Studio Code anlegen

Da das SAP Business Application Studio auch auf VS Code basiert, funktioniert die lokale Variante analog zur Cloud-Variante. Dazu nutzen wir den *SAP Fiori Application Generator* aus den zuvor installierten SAP Fiori Tools.

Über die Tastenkombination Ctrl + ⇧ + P bzw. über den Menüpfad **Anzeigen • Befehlspalette** öffnen Sie die Befehlspalette (siehe Abbildung 5.32).

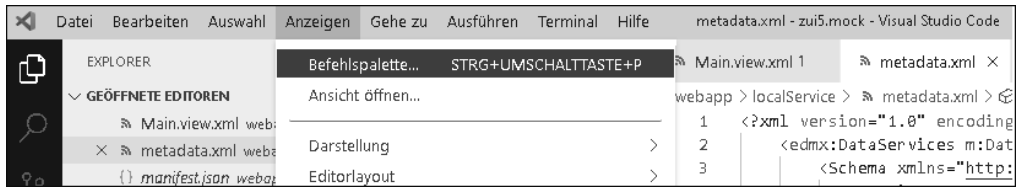

Abbildung 5.32 Command-Palette in VS Code

Tippen Sie im darauffolgenden Fenster »Application« ein, und wählen Sie dann die Option **Fiori: Open Application Generator** aus (siehe Abbildung 5.33).

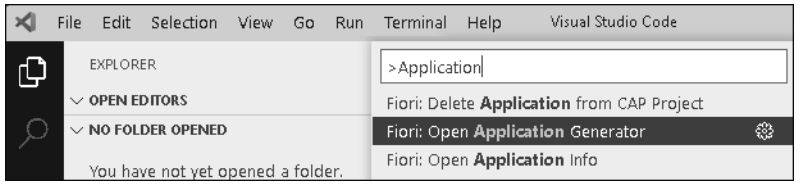

Abbildung 5.33 SAP Fiori Application Generator öffnen

Es öffnet sich der gleiche Wizard wie im SAP Business Application Studio (siehe Abbildung 5.16). Wählen Sie auch hier **SAPUI5 freestyle** als Typ aus und die Option **None** im Feld **Data Source**. Analog zur Anlage im SAP Business Application Studio nennen Sie auch hier den ersten View »Main« (siehe Abbildung 5.15).

Beim letzten Schritt, der Angabe des Speicherortes für das Projekt, empfehle ich Ihnen, ein zentrales Verzeichnis anzulegen, in dem Sie alle Ihre SAPUI5-Projekte verwalten. Ich habe dieses Verzeichnis bei mir **git** genannt (siehe Abbildung 5.34), da ich für alle Projekte ein Git-Repository nutze.

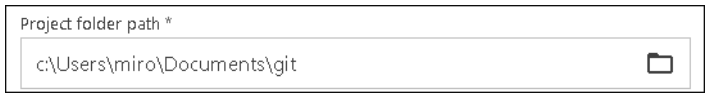

Abbildung 5.34 Projektverzeichnis anlegen

Über die weiteren Einstellungen wie **Deployment Configuration** etc. (siehe auch Abbildung 5.44) machen wir uns zu gegebener Zeit Gedanken.

Nachdem Sie auf **Finish** geklickt haben, wird das entsprechende Projekt erzeugt. Warten Sie, bis die Pakete installiert sind (siehe Abbildung 5.35), um das entsprechende Projekt zu öffnen.

Abbildung 5.35 Installation der Node-Pakete

Technisch wird neben der Verzeichnisstruktur und vielen anderen Dateien die Datei **package.json** erzeugt und die unter dem Abschnitt *DevDependencies* angegebenen Pakete installiert.

> **Git**
>
> Git ist ein Tool zur verteilten Versionsverwaltung von Dateien. Im Gegensatz zu ABAP arbeiten Sie bei der Entwicklung von SAPUI5-Anwendungen in der Cloud bzw. lokal auf Ihrem Rechner, und am Ende des Tages muss der Code bereitgestellt werden. Da Sie oft nicht allein an einer App arbeiten, benötigen Sie eine zentrale Stelle, an der Sie den Code ablegen und verwalten können. An dieser Stelle ist der Einsatz von Git also sinnvoll. Ein Git-Repository liefert Ihnen noch einen weiteren Vorteil, es fügt Dateiänderungen von verschiedenen Personen zusammen (Funktion *merge*), da im Gegensatz zu ABAP keine Sperre auf die Datei gesetzt wird.

5.3.6 Anwendung in Visual Studio Code ausführen

Analog zum SAP Business Application Studio können Sie die Anwendung über das Kontextmenü starten und ein `npm`-Skript auswählen oder eine Run-Konfiguration anlegen (siehe Abbildung 5.36).

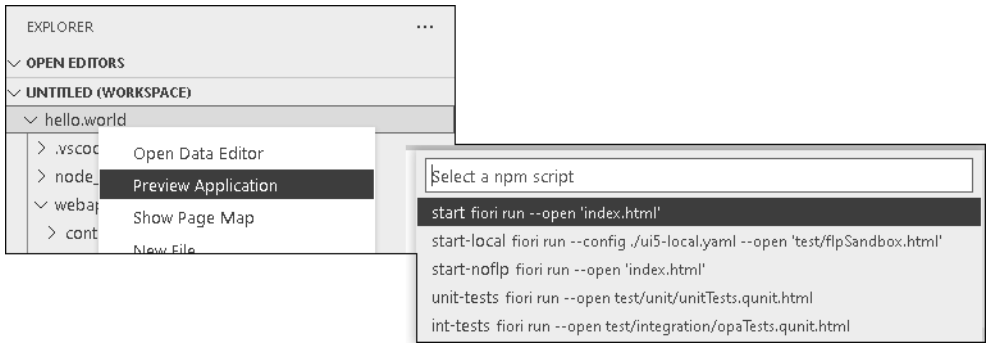

Abbildung 5.36 Anwendung in VS Code ausführen

Alternativ gibt es auch hier die Möglichkeit, eine Konfiguration anzulegen, wie bereits in Abschnitt 5.3.2, »Projekt öffnen«, beschrieben.

5.3.7 Vorhandenes Projekt in Visual Studio Code importieren

Die Beispiele aus dem Buch können Sie selbstverständlich auch lokal ausführen. Im Unterschied zum SAP Business Application Studio müssen Sie das Projekt aber vorher entpacken. Entpacken Sie dazu die entsprechende ZIP-Datei, und öffnen Sie das Verzeichnis in VS Code über das Menü **File • Open Folder**, wie in Abbildung 5.37 dargestellt.

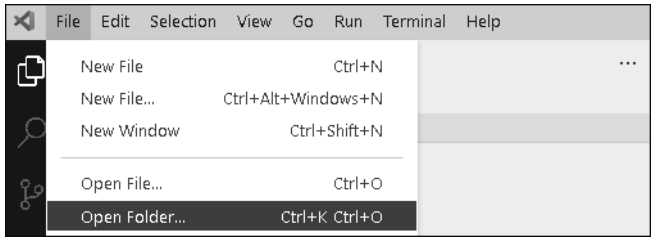

Abbildung 5.37 Projekt importieren

Wie im SAP Business Application Studio müssen Sie auch hier die `npm`-Pakete installieren, damit Sie das Beispiel ausführen können. Gehen Sie dazu mit der rechten Maustaste auf das Projekt, und wählen Sie, wie Sie es aus Abschnitt 5.3.4, »Vorhandenes Projekt ins SAP Business Application Studio importieren«, kennen, **Open in Terminal** aus (siehe Abbildung 5.30), um das Terminalfenster zu öffnen. Führen Sie dort den Befehl **npm install** aus, wie schon in Abbildung 5.31 dargestellt.

5.4 SAPUI5-Demokit

Bevor wir nun mit der Implementierung unserer ersten SAPUI5-Anwendungen beginnen, werfen wir noch einen Blick in das SAPUI5-Demokit, das Sie im SAPUI5 Software Development Kit (SDK) über die URL *https://sapui5.hana.ondemand.com* erreichen. Im Bereich **Samples** finden Sie dort Beispiele aus der SAPUI5-Bibliothek. Unter **API Reference** finden Sie die vollständige Dokumentation aller Namensräume.

Gerade am Anfang der Arbeit mit SAPUI5 hat es sich bewährt, den Samples-Bereich zu durchstöbern (siehe Abbildung 5.38), da Sie dort Anwendungsbeispiele der gängigen Controls finden.

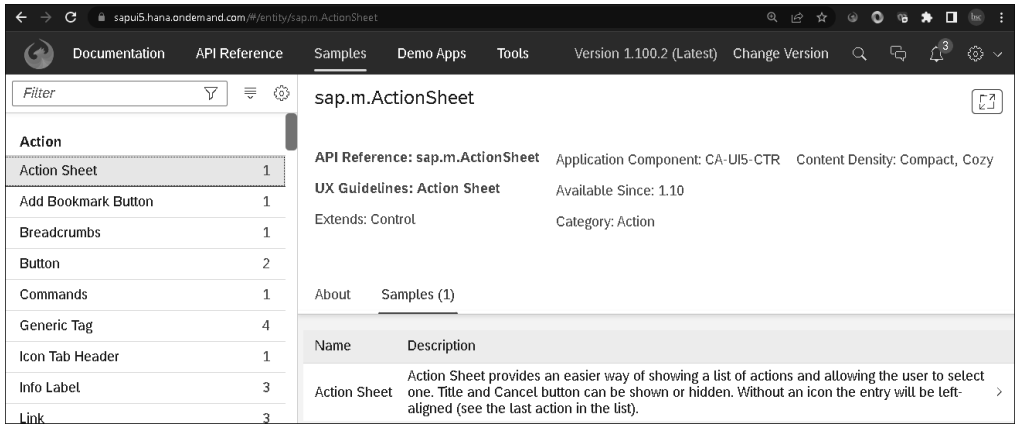

Abbildung 5.38 Bereich »Samples« im SAPUI5-Demokit

Auf der linken Seite sehen Sie die Liste der User-Interface-(UI-)Controls. Wählen Sie hier einen Eintrag aus, wird Ihnen auf der rechten Seite eine Beschreibung des entsprechenden Controls angezeigt. Im Kopfbereich finden Sie einen Link zur API-Referenz, dort finden Sie alle Informationen zu den Eigenschaften (**Properties**), Aggregationen, Ereignissen und Methoden. Auf der Registerkarte **Samples** finden Sie ein oder mehrere Beispiele zur Verwendung des Controls. Wählen Sie einen Eintrag aus der Liste der Beispiele aus, wird zunächst eine mögliche Visualisierung angezeigt (siehe Abbildung 5.39).

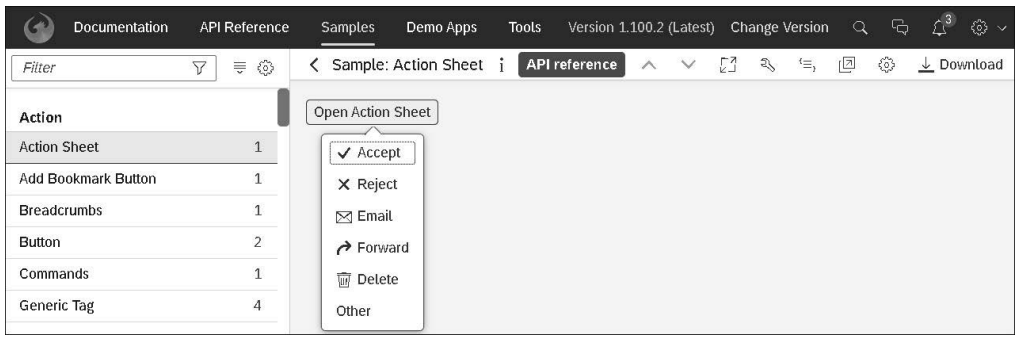

Abbildung 5.39 Visualisierungsbeispiel eines Controls

Besonders praktisch ist der Button **Show Source Code** (</>) oben rechts. Über diesen Button können Sie direkt in die Quellcodeansicht wechseln (siehe Abbildung 5.40) und das Beispiel bei Bedarf auch herunterladen.

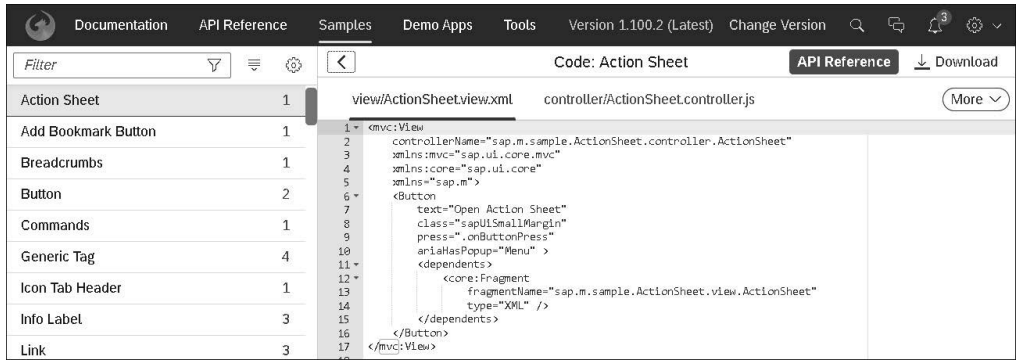

Abbildung 5.40 Quellcodeansicht für ein Beispiel-Control

Entdeckungstour im Demokit

Holen Sie sich am besten eine Tasse Kaffee oder Tee, und »surfen« Sie sich durch alle Beispiele des Demokits. Nur so bekommen Sie einen guten Überblick über die verfügbaren Controls. Wenn Sie ein Beispiel aus diesem Buch nachprogrammieren, sollten Sie dabei auch immer das SAPUI5 SDK geöffnet haben und sich die entsprechende Dokumentation zu den Controls anschauen. So erfahren Sie beispielsweise, welche Properties ein Control noch bietet. Viele Controls werden Ihnen in den folgenden Beispielen begegnen. Schauen Sie daher auch einmal im Index dieses Buches unter dem Stichwort »Control« nach, wenn Sie gezielt nach Einsatzbeispielen zu einem Control suchen.

Die schnellsten Lernerfolge erzielen Sie, wenn Sie mit den Beispielen im Buch etwas experimentieren. Probieren Sie z. B. die verschiedenen Properties aus, und betrachten Sie auch die Vererbungshierarchie. Das Control `ActionSheet` in Abbildung 5.38 erbt beispielsweise von dem Control `sap.ui.core`, dessen Properties und Methoden Sie im SDK einsehen können. Dieses Control erbt wiederum von `sap.ui.Element`, für das Sie die Aggregation `Tooltip` finden. Das bedeutet, dass Sie für ein `ActionSheet`-Control einen Tooltip implementieren können.

5.5 Ihre ersten SAPUI5-Anwendungen

Jetzt ist es an der Zeit, die Theorie in die Praxis umzusetzen. Sie legen nun Ihre erste SAPUI5-Anwendung an. Alle Beispiele in diesem Buch basieren auf einem lokal installierten Visual Studio Code. Wie Sie bereits in Abschnitt 4.2, »Visual Studio Code«, gesehen haben, verhalten sich Visual Studio Code und SAP Business Application Studio nahezu identisch.

5.5.1 SAPUI5-Anwendung entwickeln

Starten Sie als Erstes den Wizard zum Anlegen einer App. Um ein neues Projekt im SAP Business Application Studio anzulegen, gehen Sie entweder über das Menü **Project • New Project from Template**, oder klicken Sie im Willkommensbildschirm auf den Button **Start from Template**.

Im Visual Studio Code starten Sie die Befehlspalette über die Tastenkombination Ctrl + ⇧ + P bzw. über das Menü **Anzeigen • Befehlspalette** und starten den **Fiori Application Generator**. Im ersten Schritt des Wizards können Sie entscheiden, welchen Applikationstyp Sie anlegen wollen. Zur Auswahl stehen Ihnen SAP Fiori Elements und SAPUI5 Freestyle. SAP Fiori Elements ist ein Framework, das die am häufigsten verwendeten Floorplans für typische Anwendungen, wie z. B. List Reports, umfasst. Sie basieren auf den Annotationen der Core Data Services. Bei den SAPUI5-Freestyle-Templates gibt es je ein Template für eine Worklist-Anwendung, eine Master-Detail-Anwendung oder ein leeres SAPUI5-Anwendungs-Template (siehe Abbildung 5.41).

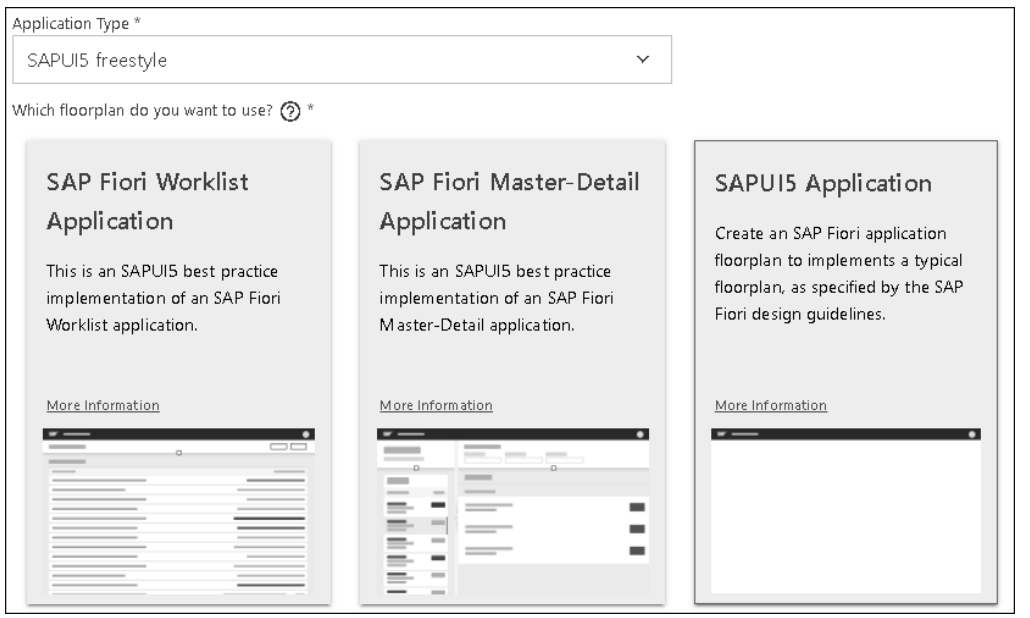

Abbildung 5.41 Template-Auswahl im Wizard

Das Master-Detail-Template wird Ihnen in Kapitel 10, »Beispielentwicklung einer SAP-Fiori-App«, noch einmal begegnen. Jetzt wählen Sie wie bisher die Option **SAPUI5 Application** aus. Natürlich könnten Sie bei der Implementierung der Beispiele mit Tabellen das Worklist-Template verwenden, meiner Meinung nach lernen Sie es aber

besser, wenn Sie eine Tabelle erst einmal selbst programmieren. Dann verstehen Sie, was das Template macht, und können es bei Bedarf anpassen. Wählen Sie also **SAPUI5 Application** aus, und klicken Sie auf **Next**. Im nächsten Schritt können Sie den Datenlieferanten wählen (siehe Abbildung 5.42).

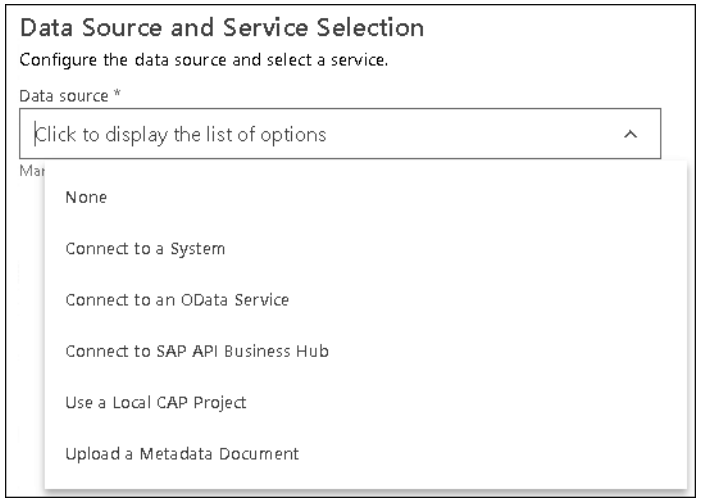

Abbildung 5.42 Mögliche Datenquellen

Zur Auswahl stehen Ihnen die Option **None**, die Sie immer dann wählen, wenn Sie keine Datenquelle anbinden möchten. Mit der Option **Connect to a System** können Sie sich mit einem ABAP-System on premise oder in der Cloud verbinden. Mit der Option **Connect to an OData Service** stellen Sie eine direkte Verbindung zu einem OData-Service her, diese Variante werden wir in Kapitel 8, »Systemanbindung mit OData«, verwenden. Wenn Sie sich mit einem Service aus dem SAP API Business Hub (*https://api.sap.com/*) verbinden möchten, wählen Sie die Variante **Connect to SAP API Business Hub**. Wenn Sie an einem CAP-Projekt (SAP Cloud Application Programming Model, *https://cap.cloud.sap/docs/*) arbeiten, wählen Sie **Use a Local CAP Project**. Wenn Sie eine Metadata-Datei haben (siehe Kapitel 8), können Sie die letzte Variante **Upload a Metadata Document** wählen. Diese ist besonders dann interessant, wenn Sie über kein Backend-System verfügen und einen OData-Service mit einem *Mockserver* simulieren möchten. In Kapitel 8 stelle ich Ihnen diese Variante vor.

Für unser erstes Beispiel wählen Sie **None** und klicken auf **Next**. Im nächsten Schritt legen Sie den View-Namen fest (siehe Abbildung 5.43).

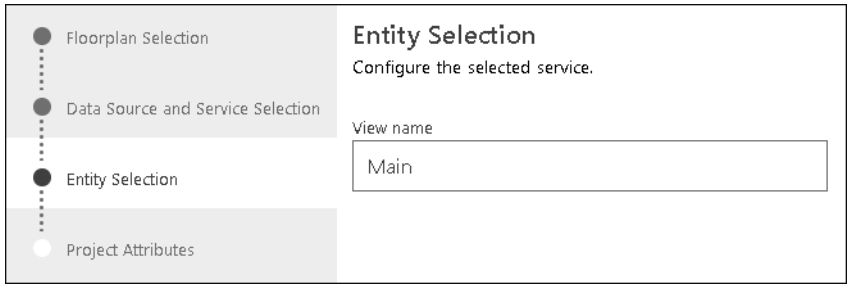

Abbildung 5.43 View-Namen festlegen

Da sich im Template bereits ein View befindet, können Sie hier den Namen angeben. Oft wählt man **Main**, **Root** oder **App**. Ich kann Ihnen gar nicht genau erklären, warum, aber ich wähle bei Apps mit nur einem View die Bezeichnung **Main** aus, bei Apps mit Navigation immer **App**. Dies ist zwar nicht besonders konsequent, aber zumindest originell. Legen Sie also einen Namen fest, und klicken Sie **Next**. Im letzten Schritt konfigurieren Sie die Projektattribute, wie z. B. den Namen und die Beschreibung (siehe Abbildung 5.44).

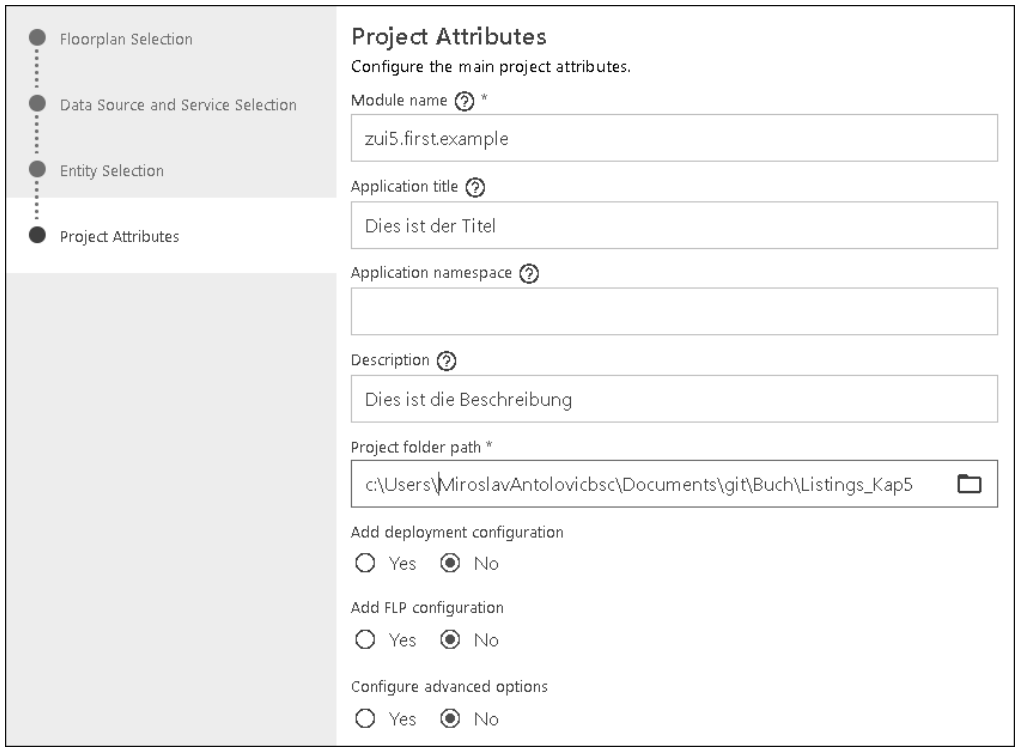

Abbildung 5.44 Projektattribute

Füllen Sie also die Felder **Module name**, **Application title** und **Description** entsprechend aus. Beim Feld **Application namespace** wird es spannender. Der Namensraum wird in SAP Fiori verwendet, um eine App eindeutig zu identifizieren. Beim Upload der App wird vom System geprüft, ob bereits eine App unter diesem Namensraum existiert, und gegebenenfalls der Upload verhindert. Sie werden in Kapitel 6, »SAPUI5-Laufzeitumgebung«, die SAP-eigenen Namensräume in SAPUI5 kennenlernen. Wenn Sie eine eigene Bibliothek entwickeln, kann diese z. B. von anderen Apps über den definierten Namensraum geladen werden. Da wir im Moment keine SAP-Fiori-App entwickeln möchten, können Sie das Feld leer lassen. Um die **deployment configuration** kümmern wir uns im nächsten Abschnitt, die **FLP configuration** (SAP Fiori Launchpad) wird erst in Kapitel 10, »Beispielentwicklung einer SAP-Fiori-App«, relevant. Die **advanced options** sind dann interessant, wenn Sie eine andere SAPUI5-Version einstellen möchten. Visual Studio Code bzw. das UI5-Tooling laufen jedoch immer auf der aktuellen Version. Wenn Ihr verwendetes Backend auf einem älteren Release läuft, sollten Sie dieses Release auch während der Entwicklung verwenden, sonst laufen Sie Gefahr, Controls oder Methoden zu verwenden, die es im Zielsystem auf dem älteren Release noch nicht gibt. In den Advanced Options können Sie auch das Theme anpassen (siehe Abbildung 5.45). Zum Zeitpunkt des Schreibens ist SAP Quartz aktuell, das neue Theme *Horizon* wurde jedoch schon angekündigt. Ich vermute, wenn Sie das Buch in den Händen halten, wird das neue Theme bereits verfügbar sein. Sie können auch ein eigenes Theme erstellen, z. B. um es Ihrem Corporate Design anzupassen. Das lernen Sie in Abschnitt 7.2.2, »UI Theme Designer«.

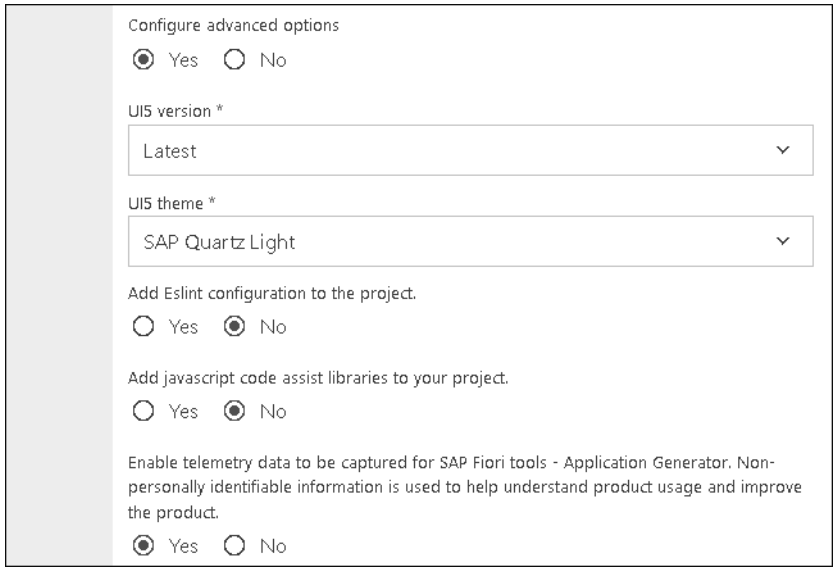

Abbildung 5.45 Advanced Options

In den Advanced Options können Sie zudem einstellen, ob Sie eine ESLint-Konfiguration hinzufügen möchten. *ESLint* ist ein Werkzeug zur statischen Quellcodeanalyse. Es identifiziert problematische Stellen und Muster in JavaScript-Codezeilen. ESLint ist Teil des UI5-Toolings. SAP liefert eine Standardkonfiguration (*http://s-prs.de/v890101*), hier können Sie bei Bedarf eine eigene Konfigurationsdatei angeben.

Wenn Sie bei der Option **Add javascript code assist libraries to your project** das Optionsfeld **Yes** auswählen, werden ein paar weitere Dateien angelegt, wie z. B. eine **.gitignore**-Datei, wenn Sie ein Git-Repository verwenden, eine **tsconfig.json**-Datei für *TypeScript* usw. Für den Einstieg ist dies aber alles nicht relevant, daher belassen Sie alles auf **No** und drücken auf **Finish**. Schauen wir uns das Ergebnis der Template-Generierung etwas genauer an (siehe Abbildung 5.46).

Abbildung 5.46 App-Struktur

Wie ich bereits erwähnt habe, ist SAPUI5 im Model-View-Controller-Architekturmuster implementiert. Wie Sie sehen, werden der View und der Controller jeweils durch eine Datei repräsentiert, der View als **Main.view.xml** ❶ und der Controller als **Main.controller.js**. Weitere Informationen zum MVC-Muster finden Sie in Abschnitt 6.1.2, »Generierung des Model-View-Controller-Musters in SAPUI5«. Die Datei **style.css** ❷ ist für Ihre individuellen Layoutanpassungen vorgesehen. Wir werden diese Datei in Abschnitt 7.2, »Layoutanpassungen«, verwenden.

Die Komponente **Component.js** ❸ mit der Datei **manifest.json** ❹ repräsentiert die Komponente, die letztlich Ihre App lädt und anzeigt. Komponenten sind unabhängige und wiederverwendbare Teile, die in SAPUI5-Anwendungen eingesetzt werden. Mehr zu den Components lernen Sie in Abschnitt 7.3, »Components«.

Die Datei **mockserver.js** ❺ implementiert den sogenannten Mockserver, mit dem Sie einen OData-Service simulieren können. Die Details dazu zeige ich Ihnen in Abschnitt 8.6, »Mit Mockdaten arbeiten«.

Im Verzeichnis **test** ❻ befinden sich die Dateien für Qunit und OPA5-Tests. *Qunit* ist ein JavaScript-Testing-Framework zum Verwalten von Unit-Tests, und *OPA5* ist ein SAPUI5-Tool, das auf QUnit und JavaScript basiert. Getestet werden können z. B. Benutzerinteraktionen, die Navigation innerhalb der App sowie das Data Binding. Auf beide Tools werde ich in diesem Buch nicht mehr näher eingehen. Die Testautomatisierung ist sicherlich ein spannendes Thema, würde aber im Rahmen einer Einführung zu weit gehen. In diesem ersten Beispiel einer App-Entwicklung möchte ich gar nicht zu sehr ins Detail gehen, vielmehr geht es darum, dass Sie das Einrichten, die Funktionsprüfung der Entwicklungsumgebung und den vollständigen Lebenszyklus einer App verstehen: vom Anlegen über die Entwicklung bis hin zur Bereitstellung auf dem ABAP-Stack.

Nachdem Sie die App erfolgreich angelegt haben, ist der nächste Schritt die Entwicklung der App. Öffnen Sie dazu die Datei **Main.view.xml**, sodass Ihnen der Inhalt der Datei im Editor angezeigt wird (siehe Abbildung 5.47).

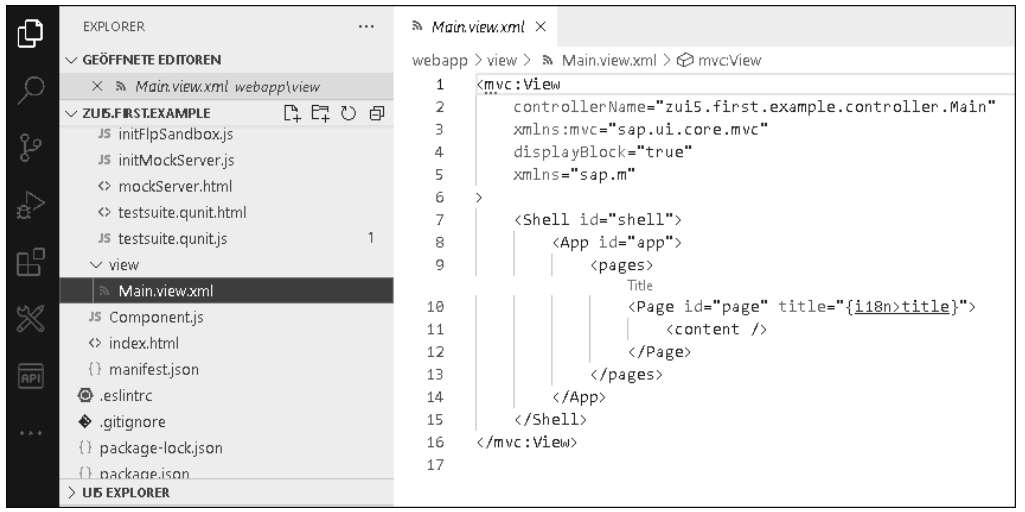

Abbildung 5.47 Main-View im Editor

Das Template hat hier bereits einiges an Code generiert. Das ist praktisch, denn so müssen Sie diesen Teil nicht mehr tippen. Schlecht daran ist nur, dass Sie als Einsteiger zunächst einmal völlig überfordert sind. Die Datei ist in XML-Notation geschrieben, schauen wir uns also die Regeln von XML an:

- **XML-Dateien haben immer einen Root-Knoten**
  Der Root-Knoten fungiert als Vater-Element, und alle weiteren Elemente (*Kind-*

Elemente) stehen unterhalb des Root-Knotens. In diesem Beispiel hat der erste Knoten den Namen `View`.

- **Alle Elemente müssen geschlossen werden**
  Die Elemente werden entweder durch ein Closing-Tag (`<View>...</View>`) oder innerhalb des Tags als Abschluss geschlossen, wie in dem Beispiel `<content />`.
- **XML-Elemente müssen ordnungsgemäß verschachtelt sein**
  In unserem Beispiel wird das `View`-Element geöffnet, innerhalb des `View`-Elements das `App`-Element geöffnet usw., dann wird das `App`-Element geschlossen und zum Schluss das `View`-Element.
- **XML-Attributwerte müssen immer in Anführungszeichen gesetzt werden**
  Das heißt, dass in dem Beispiel `<Shell id="ShellID">` das Attribut `id` des Shell-Elements den Wert `"ShellID"` erhält.
- **Elementnamen werden vom Entwickler definiert**
  Dies führt häufig zu einem Konflikt, wenn versucht wird, XML-Dokumente aus verschiedenen XML-Anwendungen zusammenzuführen. Um dieses Problem zu lösen, existieren die sogenannten Namensräume. Bei der Verwendung von Präfixen in XML muss ein Namensraum für das Präfix definiert werden. Der Namensraum kann durch ein `xmlns`-Attribut im Start-Tag eines Elements definiert werden. In unserem Beispiel wird am `View`-Element z. B. der Namensraum `mvc` (`xmlns:mvc=...`) definiert. Das `View`-Element stammt aus dem Namensraum `mvc`, deswegen ist der Knoten auch mit `mvc:View...` deklariert.

Auch SAPUI5-Elemente sind in verschiedenen Namensräumen organisiert. Deshalb müssen Sie bei der Verwendung der SAPUI5-Elemente den Namensraum XML-konform angeben. Die genaue Bedeutung der Namensräume behandeln wir in Kapitel 6, »SAPUI5-Laufzeitumgebung«.

Nachdem ich Ihnen einige Regeln von XML erklärt habe, möchte ich mit diesem Wissen unserer SAPUI5-Anwendung Elemente hinzufügen. Welche Elemente es gibt, die sogenannten *Controls*, und wo diese herkommen, werden wir ebenfalls in Kapitel 6 klären. Für die Entwicklung Ihrer ersten App müssen Sie zunächst nur wissen, dass es ein Control `Text` gibt, mit dem man Text auf der Webseite anzeigen lassen kann. Im Folgenden möchte ich Ihnen zeigen, wie Sie dieses Control in Ihre Anwendung einbinden.

Betrachten wir den Code aus Abbildung 5.47 noch einmal genauer. An welcher Stelle in Listing 5.2 könnten Sie das `Text`-Control einbauen?

```
<mvc:View
    controllerName="zui5.first.example.controller.Main"
    xmlns:mvc="sap.ui.core.mvc"
    displayBlock="true"
    xmlns="sap.m">
```

```
  <Shell id="shell">
    <App id="app">
      <pages>
        <Page id="page" title="{i18n>title}">
          <content />
        </Page>
      </pages>
    </App>
  </Shell>
</mvc:View>
```

Listing 5.2 Code des Main-Views

Falls Sie jetzt denken: »Na, überall!«, dann haben Sie grundsätzlich recht. Ob die App danach noch läuft, steht aber auf einem anderen Blatt. Um ein wenig zu üben, bauen Sie das `Text`-Control einmal direkt nach dem `View`-Element ein (siehe Listing 5.3).

```
<mvc:View
    controllerName="zui5.first.example.controller.Main"
    xmlns:mvc="sap.ui.core.mvc"
    displayBlock="true"
    xmlns="sap.m">

  <Text text="Dies ist meine erste App" />

  <Shell id="shell">
    <App id="app">
      <pages>
        <Page id="page" title="{i18n>title}">
          <content />
        </Page>
      </pages>
    </App>
  </Shell>
</mvc:View>
```

Listing 5.3 Text-Control nach dem View-Element

Sie haben alle Regeln von XML beachtet: Sie haben das Element unterhalb des Vater-Knotens positioniert, das Attribut in Anführungszeichen gesetzt und das Element geschlossen. Was soll da schon schiefgehen? Starten Sie die App, wie in Abschnitt 5.3.2, »Projekt öffnen«, bzw. Abschnitt 5.3.6, »Anwendung in Visual Studio Code ausführen«, beschrieben. Wie Sie in Abbildung 5.48 sehen, wird der Text aus dem `Text`-Control tatsächlich angezeigt.

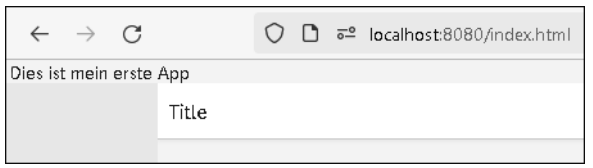

Abbildung 5.48 Erster Versuch, einen Text anzuzeigen

Der Text wird zwar richtig angezeigt, aber damit bin ich noch nicht zufrieden. Was passiert, wenn wir den Text unterhalb des Shell-Controls implementieren, also wie in Listing 5.4?

```
<mvc:View
    controllerName="zui5.first.example.controller.Main"
    xmlns:mvc="sap.ui.core.mvc"
    displayBlock="true"
    xmlns="sap.m">

  <Shell id="shell">

  <Text text="Dies ist meine erste App" />
[...]
```

Listing 5.4 Text-Control unterhalb des Shell-Controls

Wenn Sie die App nun starten, wird der Text nicht mehr angezeigt. Stattdessen erhalten Sie in der Konsole eine Fehlermeldung: »Assertion failed: multiple aggregates defined for aggregation with cardinality 0..1« (siehe Abbildung 5.49). Daraus lernen wir, dass wir die Elemente nicht beliebig verschachteln können. Was erlaubt ist und was nicht, lernen Sie in Kapitel 6.

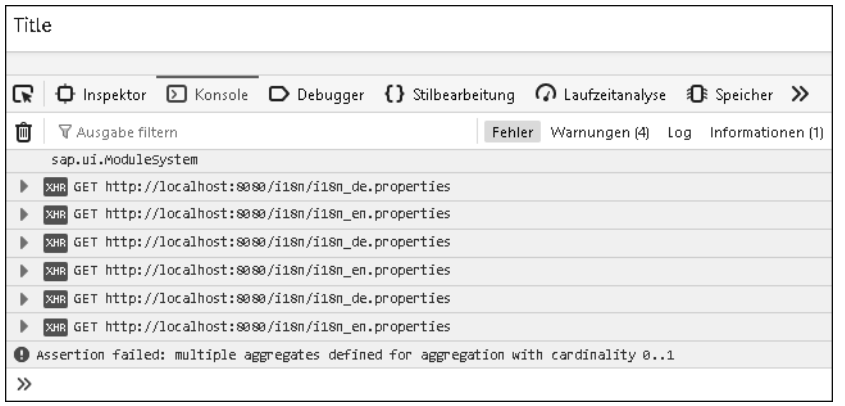

Abbildung 5.49 Fehlermeldung »Assertion failed« in der Konsole

Dieser Versuch war also nicht erfolgreich. Wir versuchen es noch mal: Fügen Sie das `Text`-Control noch einmal direkt unterhalb der App ein (siehe Listing 5.5).

```
[…]
 <Shell id="shell">
  <App id="app">
   <Text text="Dies ist meine erste App" />
   <pages>
[...]
```

Listing 5.5 Text-Control unterhalb des App-Controls

Wenn Sie die App nun starten, sollte die Seite wie in Abbildung 5.50 aussehen. Auch bei dieser Positionierung des `Text`-Controls wird der Text also angezeigt, dafür aber alles andere nicht mehr.

Abbildung 5.50 Text unterhalb des App-Controls

Das Einfügen des Textes ein Element weiter unten führt zum gleichen Ergebnis. Und wie sieht es aus, wenn wir unseren Text unterhalb des Page-Controls platzieren, wie in Listing 5.6 zu sehen?

```
<mvc:View
    controllerName="zui5.first.example.controller.Main"
    xmlns:mvc="sap.ui.core.mvc"
    displayBlock="true"
    xmlns="sap.m">

  <Shell id="shell">
    <App id="app">
      <pages>
        <Page id="page" title="{i18n>title}">
            <Text text="Dies ist meine erste App" />
        </Page>
      </pages>
    </App>
  </Shell>
</mvc:View>
```

Listing 5.6 Text unterhalb des Page-Controls

Das Ergebnis dieses Versuchs sehen Sie in Abbildung 5.51. Mir persönlich gefällt dieses Ergebnis am besten.

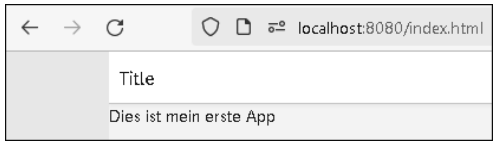

Abbildung 5.51 Der Text wird im Page-Control angezeigt.

Was lernen Sie nun aus diesem kleinen Experiment? Sie können in SAPUI5 zwar alles ausprobieren, doch nicht alles funktioniert auch. Dabei liegt es aber äußerst selten an Ihnen, wenn etwas mal nicht funktioniert, sondern eher am SAPUI5-Framework. Wenn es nicht funktioniert, suchen Sie in der **Konsole** nach einer Fehlermeldung und versuchen, aus dieser schlau zu werden. Aber noch viel wichtiger: Seien Sie mutig. Probieren Sie Dinge aus, so lernen Sie immer noch am schnellsten.

Ich lasse mich an dieser Stelle auf ein Experiment ein. Arbeiten Sie den Rest des Buches durch, und kehren Sie an diese Stelle zurück. Wenn Sie dann verstanden haben, was ich hier genau treibe, dann macht mich das stolz. Dann haben Sie etwas gelernt, und ich freue mich von ganzem Herzen für Sie. Falls Sie es nach dem Studium der restlichen Kapitel immer noch nicht verstanden haben, dann glauben Sie mir, es liegt nicht an Ihnen, sondern diesmal an mir. Schreiben Sie mir gerne eine E-Mail, an *m.antolovic@bsc-solutions.com*, mit Tipps, was ich in Zukunft besser machen kann.

Jetzt haben Sie für den Anfang genug programmiert, für den kompletten Lebenszyklus der Anwendung müssen Sie sie noch auf den ABAP-Stack deployen.

5.5.2 Anwendung in das Backend laden

Nachdem Sie soeben Ihre erste SAPUI5-Anwendung nach dem MVC-Muster implementiert haben, bleibt nur noch die Aufgabe, die erstellte Seite im ABAP-Backend einzuchecken. Auch diese Aufgabe ist mit etwas Vorarbeit denkbar einfach. Das Deployment des ABAP-Toolings basiert auf einem OData-Service. Bevor Sie die App hochladen können, müssen Sie den OData-Service */UI5/ABAP_REPOSITORY_SRV* aktivieren. Gehen Sie dazu im Backend-System in die Transaktion /n/IWFND/MAINT_SERVICE, und klicken Sie auf den Button **Service hinzufügen** (siehe Abbildung 5.52).

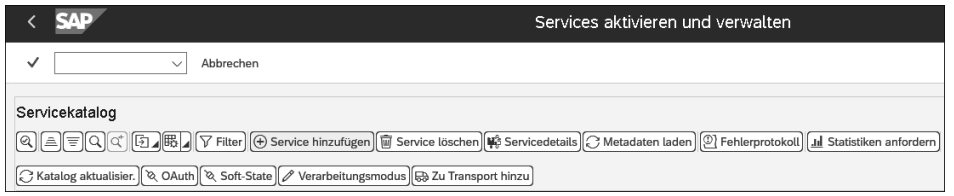

Abbildung 5.52 Service hinzufügen

Suchen Sie im Folgebild den **Systemalias** für das System aus (meist **LOCAL**) und suchen Sie nach dem Service /UI5/ABAP_REPOSITORY_SRV. Klicken Sie auf **Ausgewählte Services hinzufügn** (siehe Abbildung 5.53).

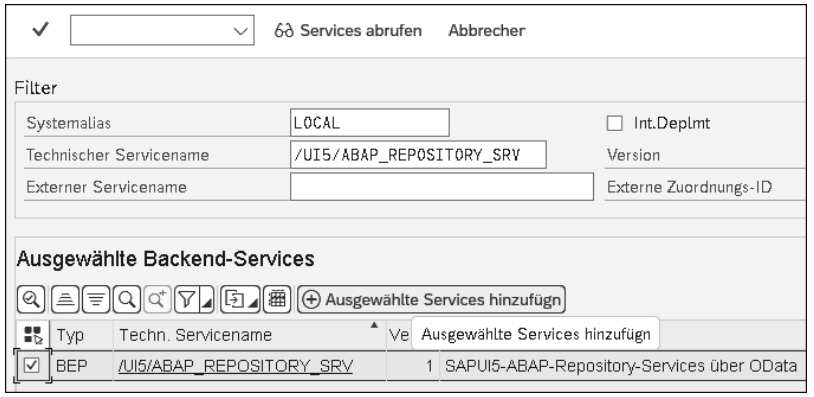

Abbildung 5.53 Service dem Servicekatalog hinzufügen

Der Service ist nicht verfügbar, was tun?

Sollten Sie auf einem älteren ABAP-Stack entwickeln, kann es durchaus vorkommen, dass der Service nicht existiert. In diesem Fall müssen Sie nach dem Deployment das **dist**-Verzeichnis manuell mit dem Report /UI5/UI5_REPOSITORY_LOAD hochladen.

Da wir beim Anlegen der App im Wizard keine Deployment-Konfiguration angelegt haben, müssen Sie diese jetzt noch erzeugen. Auch das ist kein Problem, das UI5-Tooling stellt auch dafür ein Skript bereit (siehe Abbildung 5.54).

Abbildung 5.54 Skript zum Anlegen einer Deployment-Konfiguration

Öffnen Sie über das Menü **Terminal • Neues Terminal** ein Terminal, und starten Sie das Skript über den Befehl `npm run deploy-config` (siehe Abbildung 5.55).

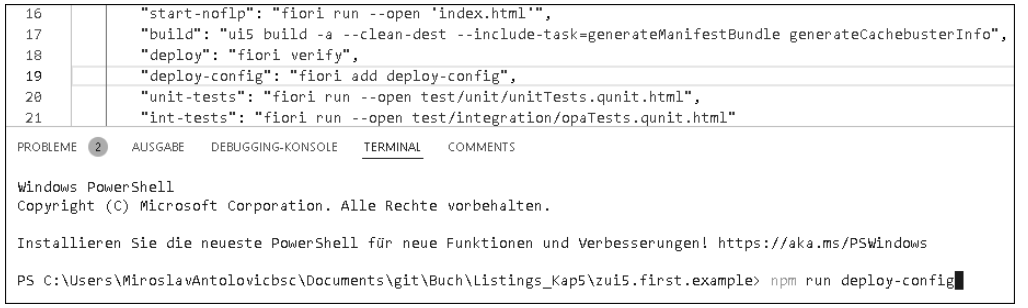

Abbildung 5.55 Deployment-Konfiguration anlegen

Das Skript startet und fragt Sie unter anderem nach dem Namen der App, dem Paket ($TMP für lokales Objekt), nach einem Transportauftrag und nach der URL des Systems. Die URL haben Sie bereits im letzten Kapitel über die Transaktion SICF ermittelt. Nachdem das Skript durchgelaufen ist, finden Sie eine neue Datei in Ihrem Projektorder, die Datei **ui5-deploy.yaml** (siehe Abbildung 5.56).

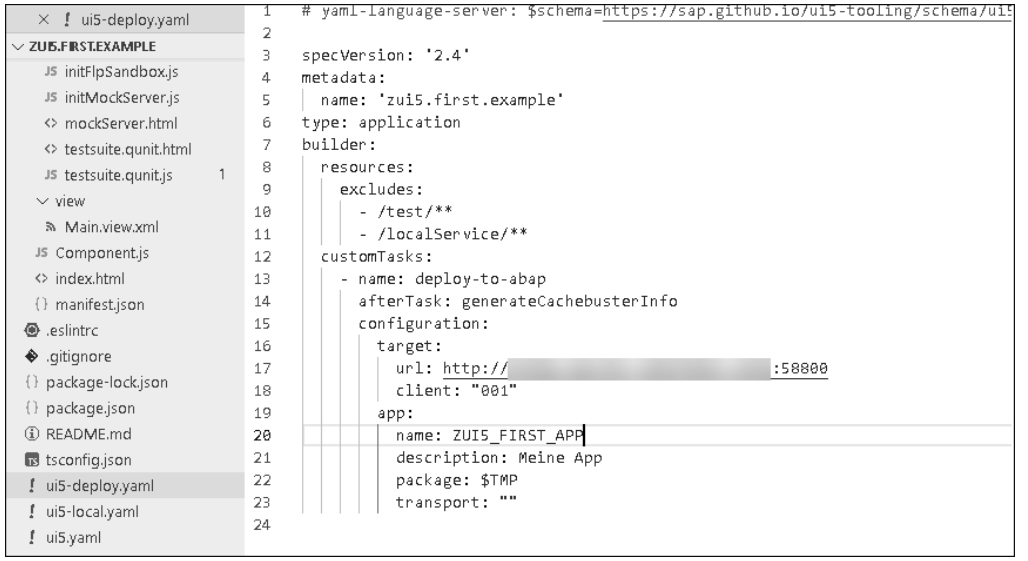

Abbildung 5.56 Datei »ui5-deploy.yaml« im Projektordner

Normalerweise legen Sie die Konfiguration direkt im Wizard beim Anlegen der App an, so werden wir es in Kapitel 10, »Beispielentwicklung einer SAP-Fiori-App«, auch machen.

Was jetzt noch fehlt, sind die Login-Daten. Die hinterlegen Sie, damit Sie sie nicht jedes Mal eingeben müssen, wenn Sie Ihre Anwendung aufrufen. In Abbildung 5.57 sehen Sie einen Ausschnitt der offiziellen Dokumentation, d. h., Sie können innerhalb des Abschnitts `configuration` einen Abschnitt `credentials` hinzufügen, um die Login-Daten zu hinterlegen. Wie Sie der Dokumentation (*http://s-prs.de/v890102*) auch entnehmen können, müssen Sie die Login-Daten in einer separaten **.env**-Datei bereitstellen.

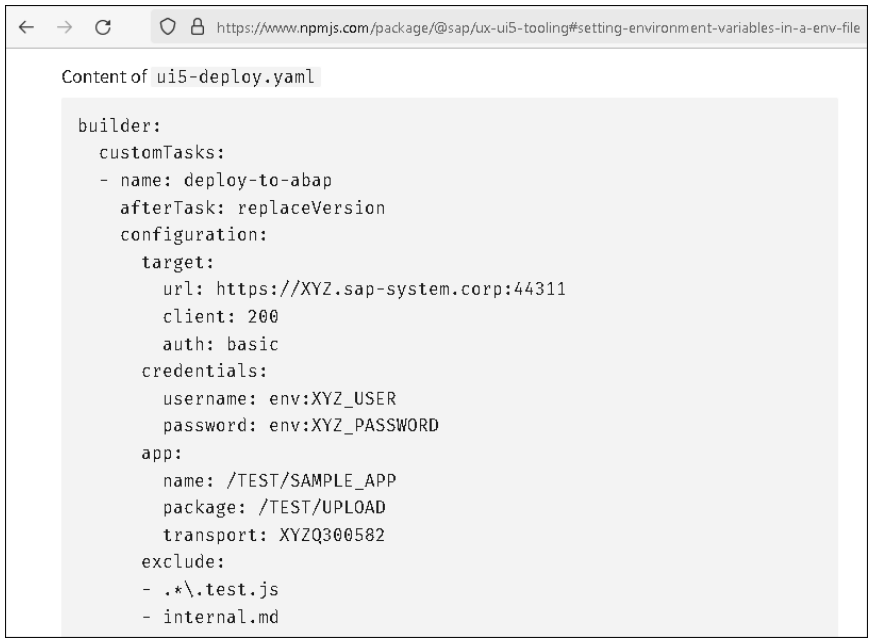

Abbildung 5.57 Erweiterung der deploy-Konfiguration um die Login-Daten

Öffnen Sie die Datei **ui5-deploy.yaml** im Editor, und erweitern Sie diese um die Zeile `credentials` mit `username` und `password`. Beim Benutzernamen schreiben Sie `env:<PLATZHALTER_NAME>` bei `password` entsprechend `env:<PLATZHALTER_PASSWORT>`. Legen Sie auf oberster Verzeichnisebene eine **.env**-Datei mit Benutzername und Passwort an (siehe Abbildung 5.58). Verwenden Sie in der **.env**-Datei die gleichen Platzhalter wie in der Datei **ui5-deploy.yaml**.

Denken Sie daran, in Ihrer Anwendung nicht meine Login-Daten einzutragen, sondern Ihren eigenen Benutzernamen und Ihr Passwort.

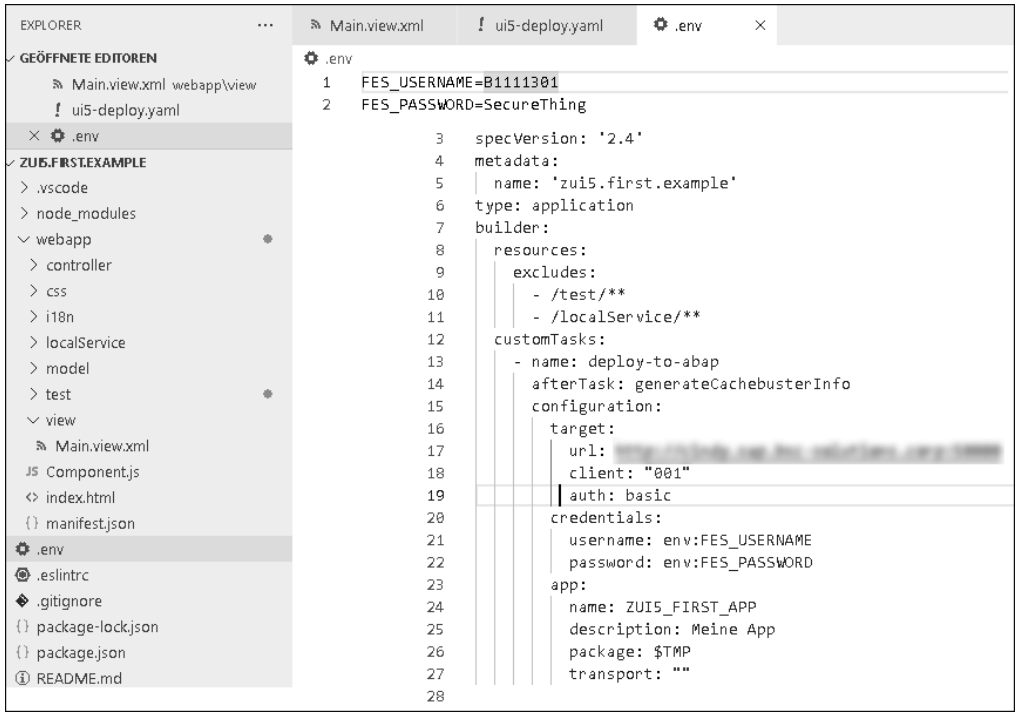

Abbildung 5.58 Login-Daten

Haben Sie dies getan, haben Sie es geschafft. Laden Sie die App ins ABAP-System hoch, indem Sie im Terminal den Befehl `npm run deploy` ausführen. Das Skript führt einige Aufgaben aus (siehe Abbildung 5.59).

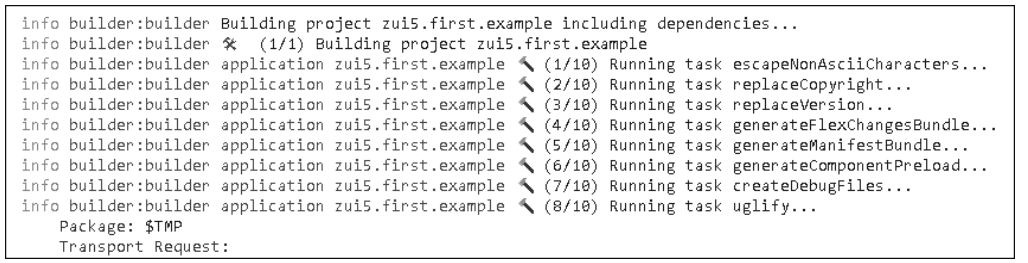

Abbildung 5.59 Deployment-Skript

Das Skript erstellt unter anderem ein neues Verzeichnis mit dem Namen **dist** und kopiert alle Dateien in dieses Verzeichnis. Es erstellt daraus die sogenannten Debug-Files (siehe auch Abschnitt 5.6.2, »Core-Komponenten debuggen«) und die **minimified**-Dateien. Das ist in der Webentwicklung üblich, da durch das Entfernen der Leer-

räume und das Kürzen der Variablennamen die Dateigröße verkleinert wird. Das entlastet den Network-Traffic und verbessert die Performance deutlich. In Abbildung 5.60 sehen Sie die beiden Dateien einander gegenübergestellt.

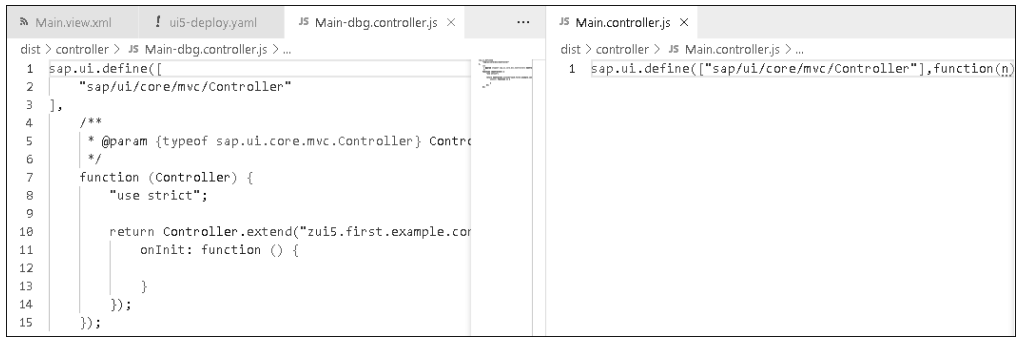

Abbildung 5.60 Gegenüberstellung der Dateien

Wenn der Deployment-Prozess abgeschlossen ist, wird Ihnen die URL der App direkt im Terminal angezeigt (siehe Abbildung 5.61).

```
info builder:custom deploy-to-abap      scmRevision: ,
info builder:custom deploy-to-abap      libraries: []
info builder:custom deploy-to-abap }
info builder:custom deploy-to-abap ... 40  messages ommited ...
info builder:custom deploy-to-abap * Updating the Application Index *
info builder:custom deploy-to-abap SAPUI5 Application  has been uploaded and registered successfully
info builder:custom deploy-to-abap * Done *
info builder:custom deploy-to-abap App available at http://[redacted]:58800/sap/bc/ui5_ui5/sap/zui5_first_app?sap-client=001
info builder:custom deploy-to-abap Deployment Successful.
PS C:\Users\MiroslavAntolovicbsc\Documents\git\Buch\Listings_Kap5\zui5.first.example>
```

Abbildung 5.61 URL zur fertigen App

Wie Sie die App im SAP Fiori Launchpad einbinden, lernen Sie in Kapitel 10, »Beispielentwicklung einer SAP-Fiori-App«. Damit haben Sie den kompletten Lebenszyklus der App geübt: vom Anlegen über die Entwicklung bis hin zum Hochladen in den ABAP-Stack.

5.5.3 Download und Import der Beispiele

Alle in diesem Buch gezeigten Beispiele finden Sie auch zum Download. Gehen Sie dazu auf die Seite *www.sap-press.de/5501*. Auf der Seite finden Sie den Abschnitt **Materialien**, wo Sie die Beispiele als ZIP-Datei herunterladen können. Nach dem Entpacken der Datei finden Sie einen Ordner **Listing** und in diesem wiederum, getrennt nach Kapiteln, die eigentlichen Beispiele als ZIP-Dateien. Im SAP Business Application Studio können Sie die ZIP-Datei direkt hochladen, in Visual Studio Code müssen Sie das Verzeichnis zunächst entpacken. In beiden Fällen müssen Sie anschließend über den Befehl `npm install` die notwendigen Pakete installieren. Welche `npm`-Pakete benötigt werden, steht in der Datei **package.json**. Die Datei **package.**

json ist das Herzstück jedes Node-Projekts. Sie beinhaltet wichtige Metadaten zu einem Projekt, die für die Veröffentlichung in `npm` erforderlich sind, und definiert auch funktionale Attribute eines Projekts. Zu den wichtigsten Attributen zählen die, um Abhängigkeiten zu installieren, Skripte auszuführen und den Einstiegspunkt zu einem Paket zu identifizieren.

In Listing 5.7 sehen Sie ein typisches Beispiel für eine SAPUI5-App, wie z. B. UI5 CLI (*https://sap/.github.io/ui5-tooling/pages/CLI/*). CLI steht hierbei für *Command Line Interface* und wird in diesem Beispiel in Version 2.11.1 installiert, wobei die drei Zahlen für Major.Minor.Patch stehen.

```
"devDependencies": {
  "@ui5/cli": "^2.11.1",
  "@ui5/fs": "^2.0.6",
  "@ui5/logger": "^2.0.1",
  "@sap/ux-ui5-tooling": "~1.0.6",
  "rimraf": "3.0.2"
}
```

Listing 5.7 »devDependencies« in der Datei »package.json«

Sie sehen, dass vor der Versionsangabe das Zirkumflexzeichen (^) auftaucht. Auch die Tilde (~) taucht hier auf. Beide Zeichen kennzeichnen das automatische Update des `npm`-Pakets in gewissen Grenzen:

- **Version mit Zirkumflex (^)**
  Die Version mit Zirkumflexzeichen verankert lediglich das Major-Release, Minor- und Patch-Versionen werden automatisch aktualisiert. Daher könnte `npm ^4.15.2` auch die Versionen 4.15.3, 4.15.4 oder 4.16.0 installieren – nicht aber die Version 5.0.0.
- **Version mit Tilde (~)**
  Die Version mit Tilde verankert Major- und Minor-Versionen, die Patch-Versionen werden automatisch aktualisiert. Hier könnte `npm` anstelle von `~4.15.2` auch die Versionen 4.15.3 oder 4.15.4 installieren, nicht jedoch 4.16.0, und schon gar nicht 5.0.0.

In beiden Fällen wird der Releasewechsel auf das nächste Major-Release nicht gemacht. Da ich den Download nicht mehr ändern kann, kann es also durchaus vorkommen, dass hier veraltete Versionsnummern stehen. Wenn Sie also eine Lösung importieren, überprüfen Sie die Versionen, und tragen Sie gegebenenfalls die aktuelle ein. Das geht ganz einfach, indem Sie die Version in der Datei **package.json** entfernen und die Tastenkombination Strg + Leertaste drücken. Es öffnet sich ein Kontextmenü, in dem Sie die aktuelle Version auswählen können (siehe Abbildung 5.62).

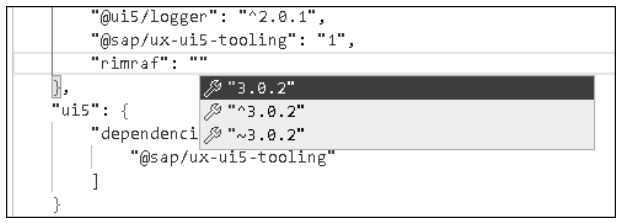

Abbildung 5.62 Version in der Datei »package.json« aktualisieren

Führen Sie anschließend wieder den Befehl `npm install` aus, um die aktualisierten Pakete zu installieren.

Alle Beispiele in diesem Buch beginnen mit `ZUI5`, der alten SAP-Tradition folgend im Z-Namensraum. Die Projekte haben als solche aber eigentlich nichts mit dem SAP-Namensraum zu tun. Wie Sie im vorangegangenen Abschnitt gesehen haben, können Sie beim Upload einen neuen Namen für die eigentliche BSP-Anwendung vergeben. Die Namensvergabe ist sicherlich Geschmackssache, aber ich habe es ganz gerne, wenn das Projekt und die spätere BSP-Anwendung namensgleich sind. Damit Sie Ihre eigenen Beispiele mit den Musterlösungen vergleichen können, verwenden Sie am besten Ihren unternehmenseigenen Namensraum, da der Projektname eindeutig sein muss.

5.6 Debugging

In Abschnitt 2.1, »Grundlagen«, haben Sie bereits die Entwicklertools des Browsers kennengelernt. Diese benötigen Sie, um eine SAPUI5-Anwendung zu debuggen.

5.6.1 Eigene Ressourcen debuggen

Wenn Sie nun z. B. die in Abschnitt 5.5, »Ihre ersten SAPUI5-Anwendungen«, erstellte Anwendung debuggen möchten, haben Sie zwei grundsätzliche Möglichkeiten, einen Breakpoint zu setzen:

- Sie können einen Breakpoint über die JavaScript-Anweisung `debugger;` im Quellcode setzen, was der Anweisung `BREAK-POINT` in ABAP entspricht.
- In den Entwicklertools können Sie zur Laufzeit einen Breakpoint setzen. Wechseln Sie dazu in den Entwicklertools auf die entsprechende Registerkarte (Chrome: **Quellcode**, Firefox: **Debugger**, Edge: **Quellen**), und navigieren Sie auf der linken Seite zu dem entsprechenden Controller, z. B. **Main.controller.js** (siehe Abbildung 5.63). Wenn Sie nun im mittleren Bereich des Bildschirms auf eine Zeilennummer des Quellcodes klicken, wird an dieser Stelle ein Breakpoint gesetzt.