Creating an enterprise-flavoured ToDo application from scratch with the Eclipse Furo Web Stack

Version 0.1| https://github.com/theNorstroem

Foreword

Hi!

This guide is intended to simplify the introduction to the FURO Web Stack. The tutorial is based on a realistic story. You will explore:

- FURO Web Stack (https://furo.pro/)
- API Design
- Go programming language (grpc backend)
- How to expose REST via gRPC Gateway
- Flow based programming
- Web Components (custom elements, shadow DOM, HTML templates, https://developer.mozilla.org/en-US/docs/Web/Web_Components)

Getting Started

We recommend 2+ years of programming experience in JavaScript / HTML / CSS and a basic knowledge of Protocol Buffers. Experience in Go is also a plus. But don't worry, you don't have to be an expert.

Complete the free 17 lectures of A Tour Of Go2 to get familiar with the syntax and basic concepts.

Setup the project

Visit the Furo ToDo Github repository and follow the installation instructions.

https://github.com/theNorstroem/todo-management-tool

Book Intro

The book guides you along a story through the functions of Eclipse Furo.
Please check out the GitHub branch for the corresponding chapter.

A checkout is marked with the `>_ git checkout` sign.

And now meet the book's main character. Peter.

Peter has been working in a software company for 10 years and he even managed to get a stake in the company. Recently, the company got a new investor and Peter had to sell his shares. With the realised profit, he bought a modern apartment building with 8 flats and reduced his workload to 60%. This allows him to do the work on the newly acquired house in his free time.

01 | The API Contract

`>_ git checkout c01_todo_api_contract`

Everything is going well, but Peter has underestimated the amount of work involved. He writes notes everywhere about what still needs to be done. Of course, Peter sometimes loses some of these notes and thus annoys his tenants a little. Peter promises to do better and he plans to build a web application to organise the tasks that come up.

He wants to write down the use cases and use them to create an API contract. This way, he can later realise the application in a short time together with his colleague Ben.

First Service

Peter starts with the first litter of the api specification. He creates a service using the Furo Interface Definition Language, so that he can record ToDo's later. He starts with the simplest version of a **TodoService.**

```
- name: TodoService
  description: Specification of the todo related services.
  package: todoservice
  target: todoservice.proto
  methods:
   - md: 'Create: POST /todos todo.Item, todo.ItemEntity #Adds a new
todo item to the todo pod'
```

Here you can find the micro service spec: `/api/muspecs/TodoService.service.yaml`

After writing the service and the request/responses types, he starts the build and looks at the artefacts. From a really simple specification he has already

- Protocol Buffer files with services and types (/dist/protos)
- a gRPC gateway in Go (https://grpc-ecosystem.github.io/grpc-gateway/)
- an ES Module for the browser platform (/dist/env.js)
- and an OpenAPI specification from his API (/dist/openapi/all-services.swagger.json)

received.

Peter is pleased and is thinking about how to proceed.

Practice time

- open a terminal
- go to the folder /api
- start the Furo Build Environment Container
  ```
  docker run -it --rm -v $(pwd):$pwd/specs
  thenorstroem/furo-bec:v1.28.5
  ```

- start the default flow of the .furo configuration file

  ```
  フロー BEC #furo
  ```

- start the build flow

  ```
  フロー BEC #furo run build
  ```

- check the /dist folder

Facts

Platform products provide their functionality via (public) APIs; hence, the design of our APIs should be based on the API as a Product principle:

- Treat your API as product and act like a product owner
- Put yourself into the place of your customers; be an advocate for their needs
- Emphasize simplicity, comprehensibility, and usability of APIs to make them irresistible for client engineers
- Actively improve and maintain API consistency over the long term
- Make use of customer feedback and provide service level support

Study Code

Setup Furo Project, first service specification
https://github.com/theNorstroem/todo-management-tool/commit/b9035991096b9cd843a83198b023284bd764b735

Summary

Eclipse Furo offers simple, enterprise-tailored, language-independent API development. It comes with multiple sources of truth and generates border-crossing type and service definitions.

https://furo.pro/

02 | Integrate API contract into the web application

>_ `git checkout c02_integrate_api_contract`

Peter wants to show his tenants the progress as soon as possible. In order not to strain their imagination too much, Peter starts by integrating the interface into a web application. This way he can get important input at an early stage of the project. He decides to put everything in a monorepo and restructures the repository.

The new project structure now looks like this:

```
.
|-- LICENSE
|-- README.md
|-- api
|-- client
```

really simple and straightforward!

Attention. This procedure is not recommended for productive projects. It makes sense to store and version the API, the client component and the server component in separate repositories.

Before Peter can implement his ideas, he develops an application body for his web application. He is a big fan of web components and he is very happy that the Furo Web Stack uses this standard. Of course, he wants to enforce a certain code quality, so he sets up a mock server, code formatting, testing and linting.

Web Application | Register a new ToDo item

After a few days of brainstorming and programming, Peter can now launch his new ToDos web application locally. Currently he is using a simple static mock server, which is set up based on the OpenApi documentation (`/api/dist/openapi/all-services.swagger.json`).

Peter has decided to start with the entry form and then gradually add more functions. To keep an overview of the source code, he structures the "src" folder according to business object related function blocks.

```
.
|-- app-shell.js
|-- configs
|-- create
|-- main-stage.js
|-- view-404.js
|-- view-5xx.js
|-- viz.js
`-- x
```

Peter adds the most important functions during the development time in the package.json in the script section. Now he can simply run the functions with

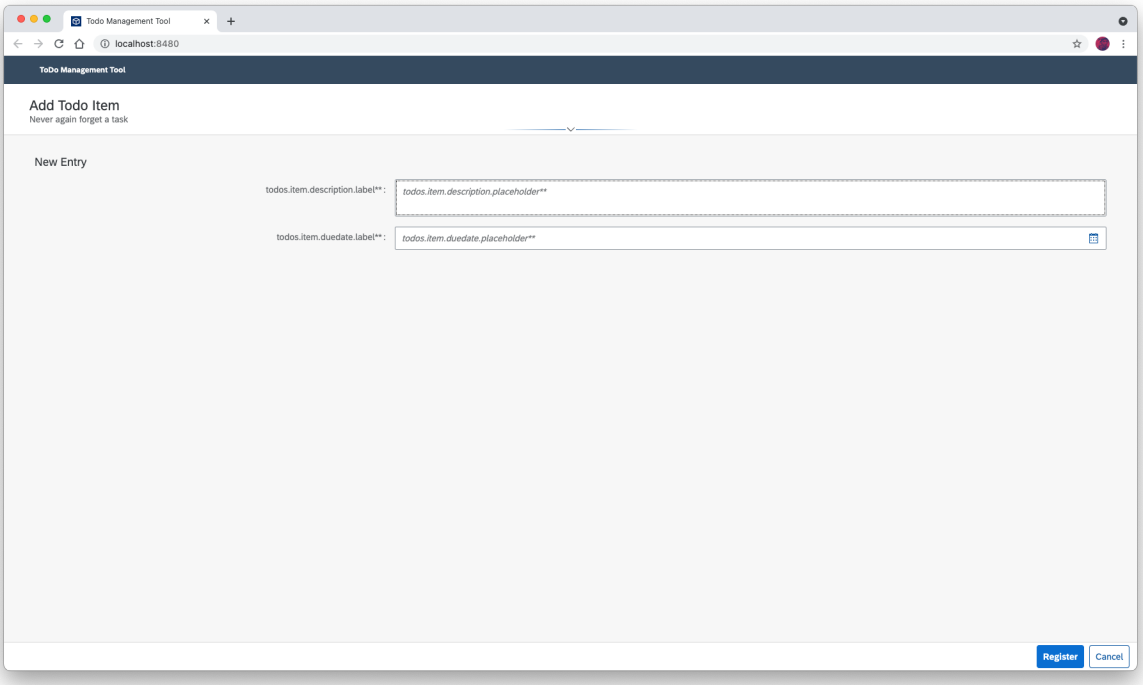
`npm run …`

- **`start:mock`** (starts the app with the mock server backend)
- **`start`** (starts the app with a backend proxy)
- **`start:build`** (starts the app from the dist folder)
- **`format`** (formats the source code)
- **`lint`** (runs eslint)
- **`test`** (runs all the tests)

To install all the listed dependencies type: `npm install`

Starting the web application locally with mock backend

- open a terminal
- go to the folder /client
- type `npm run start:mock`



`/client/src/create/view-create-todos.js`

Facts

Eclipse Furo Web is an enterprise ready set of web components which work best with Eclipse Furo. Comes with minimal footprint. Based on web standards. Future proved. Compliant with any technology of choice.

Web Components is a suite of different technologies allowing you to create reusable custom elements — with their functionality encapsulated away from the rest of your code — and utilize them in your web apps.

https://developer.mozilla.org/en-US/docs/Web/Web_Components

Summary

The complete web application is built with web components. For the easy creation of the components we used Lit, which allows the simple and fast creation of web components without boilerplate code. Lit adds just what you need to be happy and productive: reactivity, declarative templates and a handful of thoughtful features to reduce boilerplate and make your job easier. Every Lit feature is carefully designed with web platform evolution in mind.

https://lit.dev/

The following functions are currently built into the web application.

- Deep linkable web application
- Integrated api contract
- Data models
- Components for communication with the backend
- Mock server
- SAP UI5 visual lib
- Error handling
- Configurable routing
- ESLint
- Testing
- Build

Eclipse Furo Web in interaction with design systems

EclipseFuro is basically split into two modules:

EclipseFuro
Eclipse Furo offers simple, enterprise-tailored, language-independent API development. It comes with multiple sources of truth and generates border-crossing type and service definitions.

EclipseFuro-Web
Enterprise ready set of web components which work best with Eclipse Furo. Comes with minimal footprint. Based on web standards. Future proved. Compliant with any technology of choice.

Out of the box, EclipseFuro-Web comes with a UI component library based on the SAP design system Fiori. If you want to use a different design system, you can bind an existing web component library to the Furo data models with the FieldNodeAdapter. The following libraries can already be linked in this way today.

Source: https://lit.dev/

03 | Feedback Cycle, Review Process

>_ `git checkout c03_api_review`

Peter has been showing his web prototype to his tenants over the last few days and has gone through the entry process with them.

Now a few questions have arisen.

- Where are the todo items stored?
- How can I retrieve items that have already been saved?
- Is it possible to change items that have already been entered?

Packed with questions, Peter sets out to analyse the questions and works out solutions.

He quickly realises that he needs a backend system that implements his API contract. He also noticed missing endpoints in the API. He needs a list function, a get function and an update function.

Before taking care of the backend system, he extends the API specification with the missing endpoints.

- /todos with method LIST
- /todos/{tdi} with method GET
- /todos({tdi} with method PATCH

Here you can find the updated micro service spec:
`/api/muspecs/TodoService.service.yaml`

But there is a significant problem with the current implementation! **Pause reading.**
Open `/api/muspecs/TodoService.service.yaml` and try to find out the bug.

04 | Get ready for parallel development

>_ `git checkout c04_api_bugfixes`

Congratulations! You are excellent at finding bugs.

Peter adds the missing type specification for todos.ItemCollection and starts the build so that all artefacts are up to date.

```
- name: TodosService
  description: Specification of the ToDos related services.
  package: todosservice
  target: todosservice.proto
  methods:
    - md: 'Create: POST /todos todos.Item, todos.ItemEntity #Adds a
new ToDos item to the ToDos list'
    - md: 'List: GET /todos google.protobuf.Empty,
todos.ItemCollection #The List method takes zero or more parameters
as input, and returns a todos.ItemCollection of todos.ItemEntity
that match the input parameters.'
    - md: 'Get: GET /todos/{tdi} google.protobuf.Empty,
todos.ItemEntity #The Get method takes zero or more parameters, and
returns a todos.ItemEntity which contains a todos.Item'
      qp:
        tdi: 'string #The query param **tdi** stands for the item
id.'
    - md: 'Update: PATCH /todos/{tdi} todos.Item, todos.ItemEntity
#The Get method takes zero or more parameters, and returns a
todos.ItemEntity which contains a todos.Item'
      qp:
        tdi: 'string #The query param **tdi** stands for the item
id.'
```

`/api/muspecs/TodoService.services.yaml`

No sooner is the build finished than Ben calls and says he has time for frontend development in the next few days. Perfect timing. Now they can work on the project in parallel. **Many thanks to API First!**

Peter explains to Ben how the project is conceived and hands over the front-end development to him. Ben can now implement the missing functions based on the API contract.

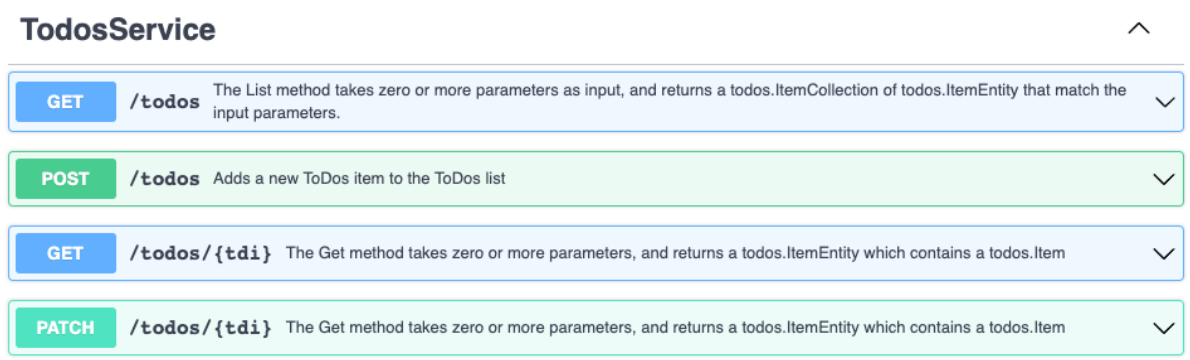
Ben is satisfied. He has the complete API specification with all defined types in his hand. As long as the backend system is not yet available, Ben works with the mock server.

Peter was kind enough to add an OpenApi 2.0 generator to the Furo Build flow.

```
// section flows:
build:
    - genBundledServiceProto
    - gen_open_api
```

`/api/.furo`

This way Ben can examine the documentation in the Swagger Editor.



https://editor.swagger.io/#

Simply copy the content of `/api/dist/openapi/all-services.swagger.json` into the left column of the editor.

05 | gRPC backend - Putting it all together

`>_ git checkout c05_grpc_backend`

Peter now has support from Ben. That comes in handy, so he can take care of the backend.

Furo provides an interface meta-description from which various artefacts can be created. Protocol Buffers, Go Structs and Go Service Interfaces are created as standard.

Of course, Peter could also write a Spring Boot backend and generate the necessary Java classes and service interfaces from the protocol buffers, but he opts for Go.

There are reasons for that. The necessary Go structs and Go services are already available and a gRPC server in Go can be set up quickly. Peter doesn't want Ben to have to wait too long for his service implementations.

First, Peter would like to change the project structure a little. The new structure should look like this.

```
.
|-- LICENSE
|-- README.md
|-- api
|-- client
`-- grpc-backend
```

After adjusting the structure, he creates a new Go module in the grpc-backend folder.

```
.
|-- cmd
|   `-- tmt-grpc
|-- data
|   `-- tmt.db
|-- go.mod
|-- go.sum
|-- internal
|   |-- env
|   |-- todo
|   `-- ulid
`-- vendor
    |-- github.com
    |-- golang.org
    |-- google.golang.org
    `-- modules.txt
```

Study Code

Look at the implementation of the gRPC server.
https://github.com/theNorstroem/todo-management-tool/blob/c05_grpc_backend/grpc-backend/cmd/tmt-grpc/grpcserver.go

Peter has placed the internal implementation of the TodosService interfaces from the module `github.com/theNorstroem/todo-management-tool/api` in a subfolder internal.

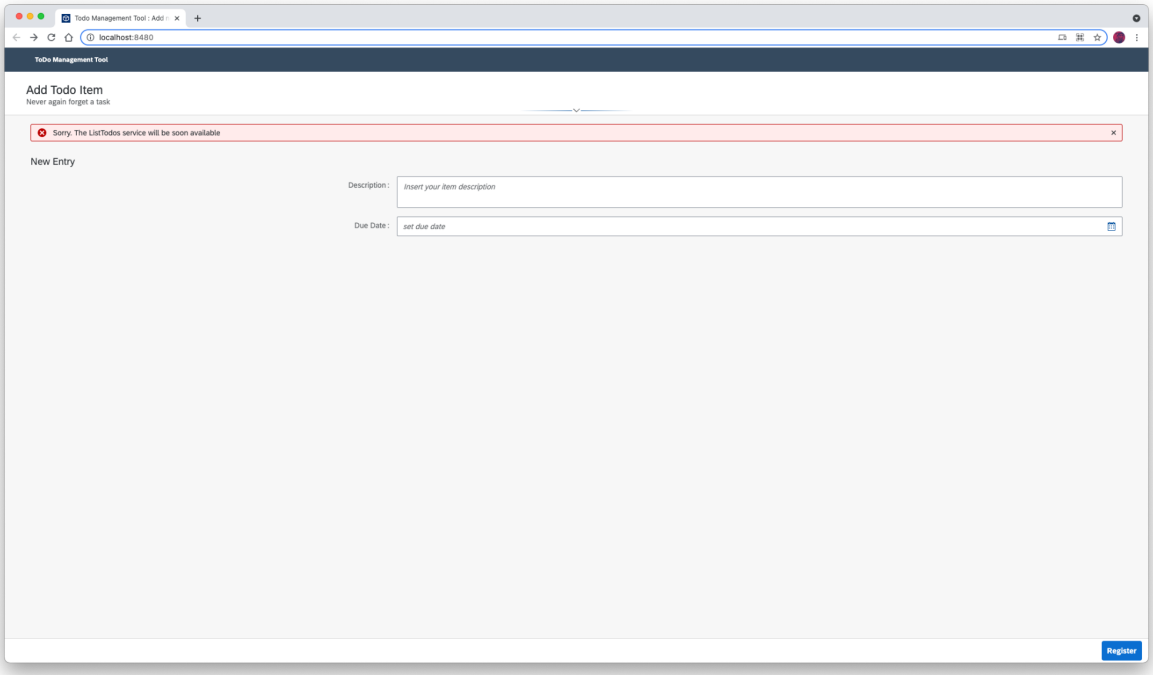
Not much is happening at the moment. On the CreateTodo endpoint, a dummy entity of the type todos.ItemEntity is simply returned. On all other endpoints, Peter implemented a google.rpc.Status with a localized message (https://cloud.google.com/apis/design/errors, https://github.com/googleapis/googleapis/blob/master/google/rpc/error_details.proto).

Should Ben already call these endpoints from the web application, he will receive a clear response.

501 Not Implemented


```
{
  "code": 12,
  "message": "This service endpoint is not yet available",
  "details": [
    {
      "@type": "type.googleapis.com/google.rpc.LocalizedMessage",
      "locale": "en-GB",
      "message": "Sorry. The ListTodos service will be soon available"
    }
  ]
}
```

In the web application it looks like this.



Starting the engine

Uff! This was a lot of information and changes. But don't worry, the following practice step will enlighten the mystery.

1. Ensure you have the latest branch's version (`c05_grpc_backend`).
2. Boot the gRPC Server in a terminal

```
cd grpc-backend
go run ./...
```

```
2021/11/10 14:04:07 server listening at 127.0.0.1:7070
```

3. Boot the gRPC Gateway

```
cd api
GW_SERVER_ADDRESS=localhost:8481
GRPC_SERVER_ADDRESS=localhost:7070 go run ./...
```

```
{"log.level":"info","log.logger":"grpc
gateway","labels.category":"APP","@timestamp":"2021-11-25T10:28:31+01:00","messag
e":"grpc gateway started on localhost:8481"}
{"log.level":"info","log.logger":"grpc
gateway","labels.category":"APP","server.address":"localhost:7070","@timestamp":"
2021-11-25T10:28:31+01:00","message":"grpc gateway connects to localhost:7070"}
```

Start the web application

```
cd client
npm run start
```

```
> todo-management-tool@1.0.0 start> es-dev-server --open

es-dev-server started on http://localhost:8480
  Serving files from
'/Users/roger.mueller/dev/theNorstroems/todo-management-tool/client'.
  Opening browser on '/'
  Using history API fallback, redirecting route requests to '/index.html'
  Using auto compatibility mode, transforming code on older browsers based on
user agent.
```

DONE! Good job.

Frontend development with the built-in components

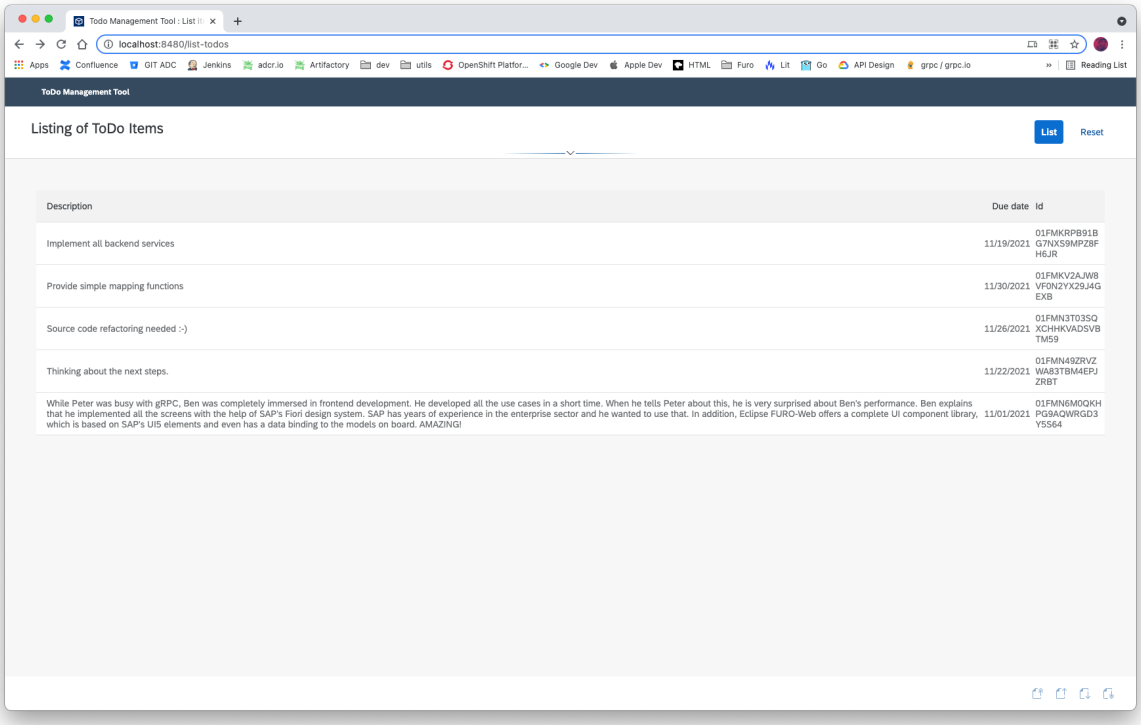
While Peter was busy with gRPC, Ben was completely immersed in frontend development. He developed all the use cases in a short time. When he tells Peter about this, he is very surprised about Ben's performance. Ben explains that he implemented all the screens with the help of SAP's Fiori design system. SAP has years of experience in the enterprise sector and he wanted to use that. In addition, Eclipse FURO-Web offers a complete UI component library, which is based on SAP's UI5 elements and even has a data binding to the models on board. **AMAZING!**

Peter and Ben are looking forward to the weekend. There comes a time when it's over. That was a lot of work.

A productive data access layer for Go

Peter needs a handful of business logic. At least a simple persistence of the data. He decides to use sqlite with upper db. This is simple and easy to expand. Peter currently stores the data locally.

Ben is overjoyed. His entry process and listing of todo items works.



Study Code

Look at the implementation of the grpc handler
https://github.com/theNorstroem/todo-management-tool/blob/cc9a8d2b9e6dbd1cb0b2f72bd2a6d4d83e070654/grpc-backend/cmd/internal/todo/todogrpc.go#L31

The sqlite database file is located on `/data/tmt.db`.

```
'017D2DB2AECEB0CC02A3466F01F2BCD8',Issue new tenancy agreements. Adapt the general conditions of use of adjoining rooms.,2021-12-31
'017D2DB3DF4F6B26DE5ADD76E164DFEB',Repair wood panelling on the 1st floor.,2021-11-30
'017D2DB691257C5B1FE737758AF63114',Prepare service charge settlement,2021-12-13
'017D2DB76D500B1E34C07558D6A34E91',Check that everything in the garden is made winter-proof.,2021-12-1
```

Facts

Generating Code based on Protocol Buffers

To generate the Java, Python, or C++ code you need to work with the message types defined in a .proto file, you need to run the protocol buffer compiler protoc on the .proto.

https://developers.google.com/protocol-buffers/docs/overview#generating

In gRPC, a client application can directly call a method on a server application on a different machine as if it were a local object, making it easier for you to create distributed applications and services. As in many RPC systems, gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. On the server side, the server implements this interface and runs a gRPC server to handle client calls. On the client side, the client has a stub (referred to as just a client in some languages) that provides the same methods as the server.

https://grpc.io/docs/languages/go/quickstart/

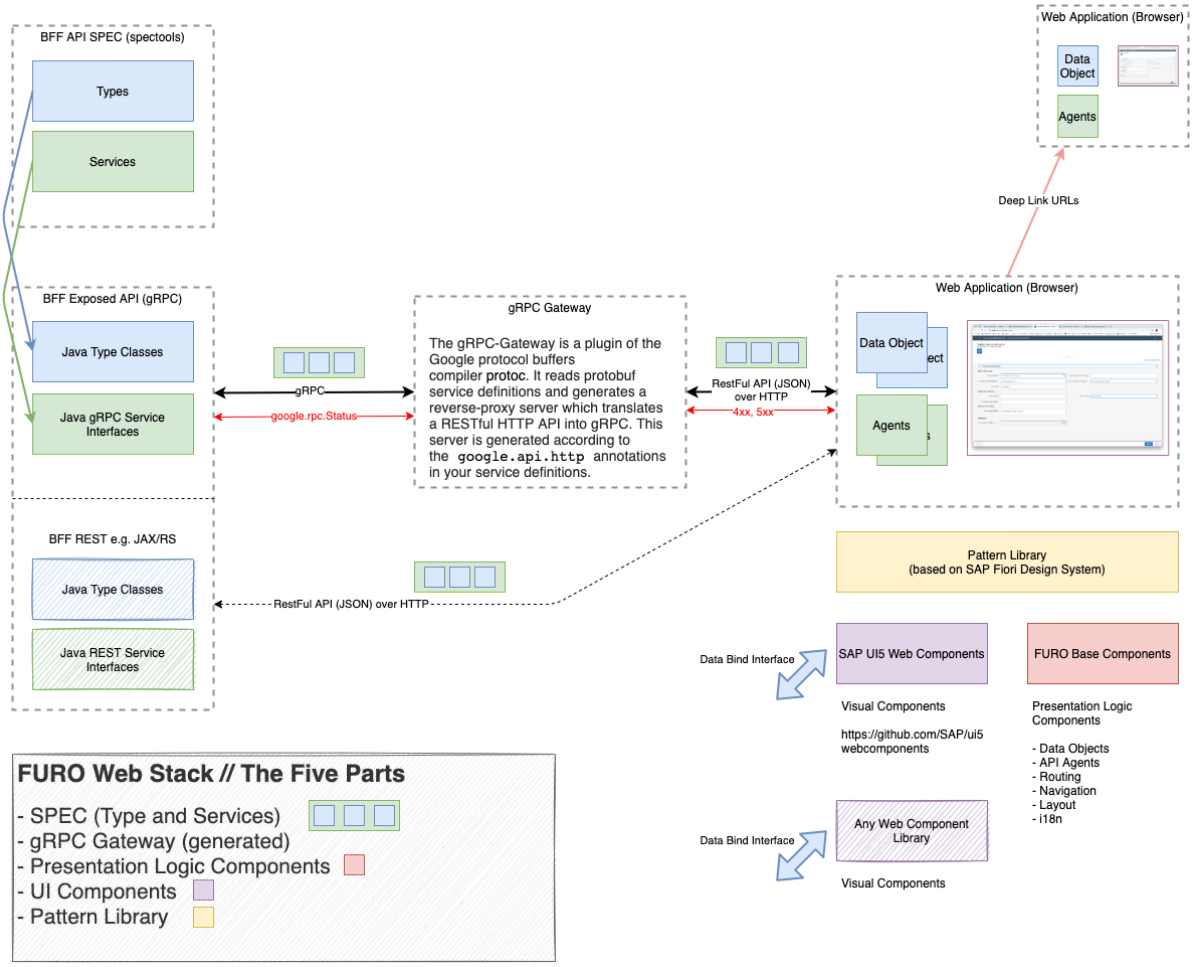
gRPC Gateway

The gRPC-Gateway is a plugin of the Google protocol buffers compiler protoc. It reads protobuf service definitions and generates a reverse-proxy server which translates a RESTful HTTP API into gRPC. This server is generated according to the google.api.http annotations in your service definitions.

This helps you provide your APIs in both gRPC and RESTful style at the same time.

https://grpc-ecosystem.github.io/grpc-gateway/

System Landscape



Glossary

Term	Description
BFF	**B**ackend **F**or **F**rontend your best friend forever :-)
gRPC Gateway	The gRPC-Gateway is a plugin of the Google protocol buffers compiler protoc. It reads protobuf service definitions and generates a reverse-proxy server which translates a RESTful HTTP API into gRPC. This server is generated according to the google.api.http annotations in your service definitions. This helps you provide your APIs in both gRPC and RESTful style at the same time.

Data Object	The furo-data-object is a web component from the package @furo/data. It provides an interface to a data type of your api specification. All the available types and services are located in the env.js (`/client/src/configs/env.js`)
Agents	Agents are web components that communicate with your backend.

06 | Minimal Viable Product

`>_ git checkout c06_mvp`

Peter wants to enter the test phase with his tenants and decides to launch an MVP.

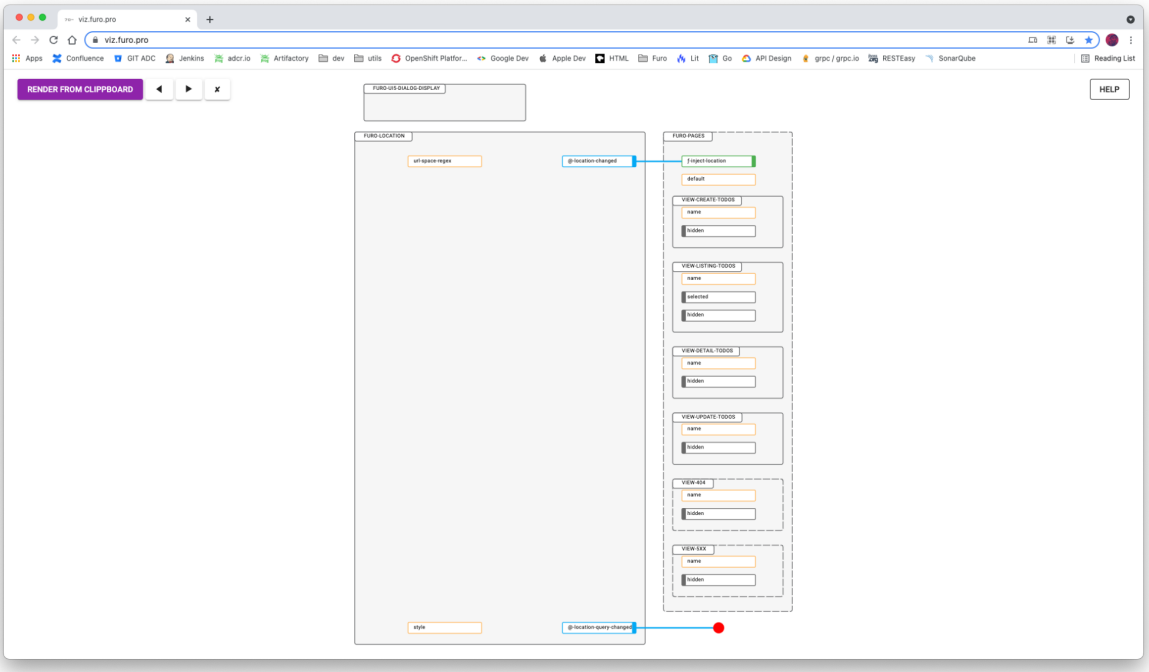
Minimum Viable Product or MVP is a development technique in which a new product is introduced in the market with basic features, but enough to get the attention of the consumers. The final product is released in the market only after getting sufficient feedback from the product's initial users.

Internal Review

Before the big day, Peter wants to go through all the use cases again. He asks Ben about the status of the front development. Ben tells him that everything is ready and that he urgently needs to show him something interesting.

Ben:
Look at this!
I created the entire web application with Furo Flow Based Programming. Now I can click through the complete application logic and see all the connections.

- open the browser's dev console
- select the desired html element
- type in the console **viz($0)** - $0 represents the selected element
- double click the element to go deeper



Peter:
That is fantastic! And how does flow-based programming work?

Ben:
Instead of registering EventListeners or selecting elements to call functions, you can simply do this with @-EVENT_NAME for events and *ƒ*-FUNCTION_NAME for functions. But there is much more. Read the documentation.

Basic principle: Pull a wire from an event to one or more functions.

```
<furo-ui5-button
      @-click="--registerRequested">Save
</furo-ui5-button>

<!-- ToDos gRPC Service -->
<furo-entity-agent
      service="TodosService"
      ƒ-hts-in="--htsOut"
      ƒ-create="--registerRequested"
      @-response="--saveOK"
      @-response-error-400="--grpcError"
      @-response-error-501="--notImplemented"
      @-fatal-error="--fatalError"
></furo-entity-agent>
```

https://github.com/theNorstroem/todo-management-tool/blob/2de996a21a7dc0e5e236d66df295eddddb2cceb2/client/src/create/view-create-todos.js#L140

Outlook

Thanks for reading! The story continues...

Topics:

- Filter / Search
- Pagination
- Object references
- Field validation
- Custom methods
- Deployment (Docker, Kubernetes)
- Testing
- Feature Toggles
- Extending a Web Component UI library with Furo data binding