

Installation Guide – Goals Application

Repositories

Service	Bereich	Link
GoalService	Backend Service	https://github.com/NikolasGoblirschUrban/goalservice.git
ItemService	Backend Service	https://github.com/NikolasGoblirschUrban/itemservice.git
UserService	Backend Service	https://github.com/NikolasGoblirschUrban/userservice.git
ConfigServer	Config Server for Backend Services	https://github.com/NikolasGoblirschUrban/configserver.git
Service Configs	Config Files for Config Server	https://github.com/NikolasGoblirschUrban/service-configs.git
Documentation	Postman-Collection, Architecture, Database Schema	https://github.com/NikolasGoblirschUrban/postman.git
GoalsBFF	BFF Gateway Application	https://github.com/NikolasGoblirschUrban/goalsbff.git
GoalsFrontend	Frontend Application	

Prerequisites

Java Development Kit

Install JDK 17 and set the required system variable.

Database Server

The database schema defined by the Spring backend services works with MySQL 8 Databases. You can either use a single database server instance per service or use the same server instance with separate databases for each service. You can install a MySQL Server on your local machine, or you can create a free subscription for any cloud provider that offers MySQL databases (e.g., Azure). After creating the database server install MySQL Workbench on your local machine to be able to connect to your server(s) and run SQL statements manually. Create the databases for the backend services and name them goals, items, and users. If you use a single server instance create all of them on the same server. Otherwise create a database named “goals” on the database server that you want to use for the goal service, etc.

Message Broker

As the backend uses an event-driven architecture a message broker is required, that provides the queues and topics that the events are sent to and consumed from. To be able to run the goals application independently from the Azure cloud the services were changed so that they use ActiveMQ. Download ActiveMQ 5.18.2 from <https://activemq.apache.org/activemq-5018002-release> and extract the zip file. Open a command shell and navigate into the extracted directory using cd. Enter the command bin\activemq start to start the message broker. The service can be stopped again using ctrl + c. You can open the administration interface by opening the following URL in a browser if the message broker is started: <http://localhost:8161/admin>. The required credentials are admin as a

username and admin as a password. You do not need to do any further configuration. The services will create the required queues and topics by themselves.

Postman

Most of the REST endpoints required for testing (e.g., simulate that a user has achieved an item) are not offered via a user interface and need to be triggered manually. Therefore, install Postman, download the Postman collection from the repository (<https://github.com/NikolasGoblirschUrban/postman.git>), and open it.

Node Package Manager and Node.js

As the Apollo server (BFF Gateway, GraphQL) is a node.js application written in typescript and angular components use typescript code the node package manager (npm) and Node.js needs to be installed.

Angular CLI

For running and developing angular applications the angular CLI needs to be installed.

Storage Account (Blob Storage)

Initially, the images required by the frontend application were stored in an Azure storage account. To be able to run the application independently from Azure this was removed, and the images were copied directly into the directory `/src/assets/images` of the frontend project. Therefore, no changes are required, and additional images can be placed into the same directory.

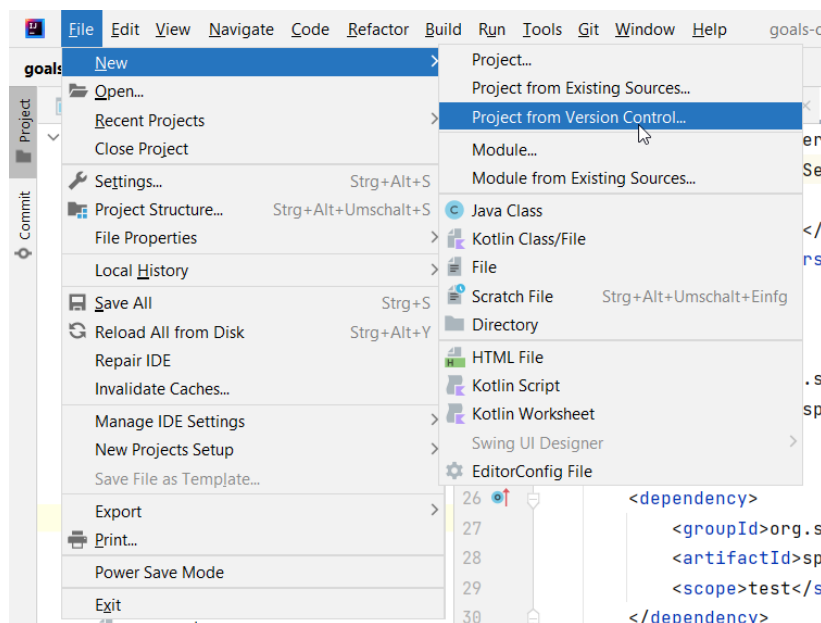
IDE

With your JKU account you can get free versions of IntelliJ and WebStorm IDEs which are highly recommended for the goals application.

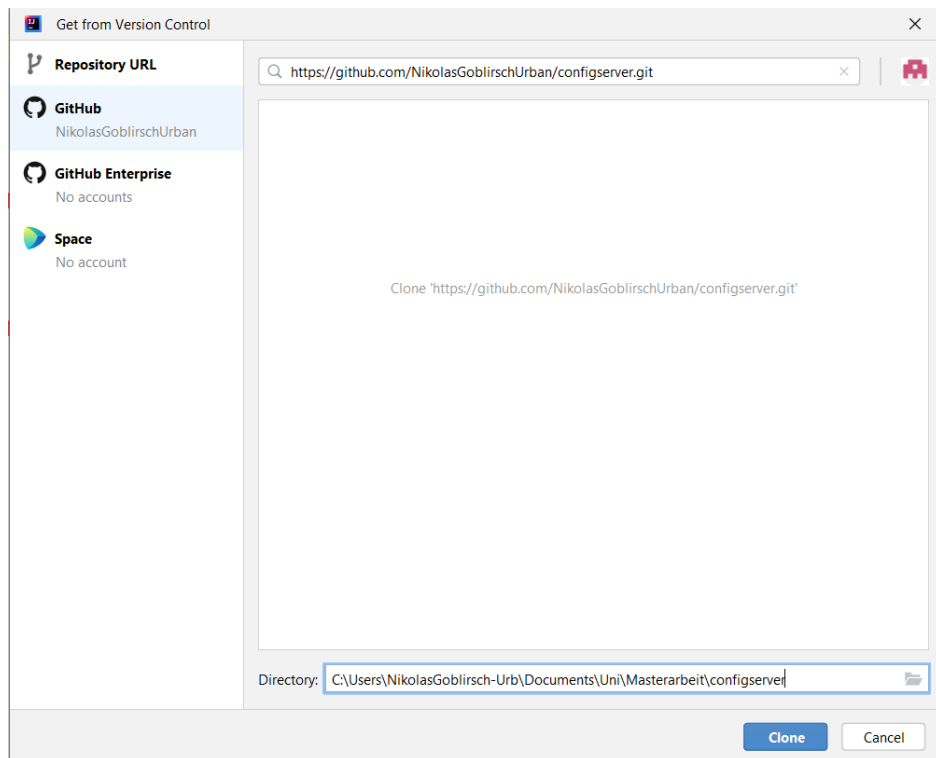
Step-by-step Guide

1. Config Server

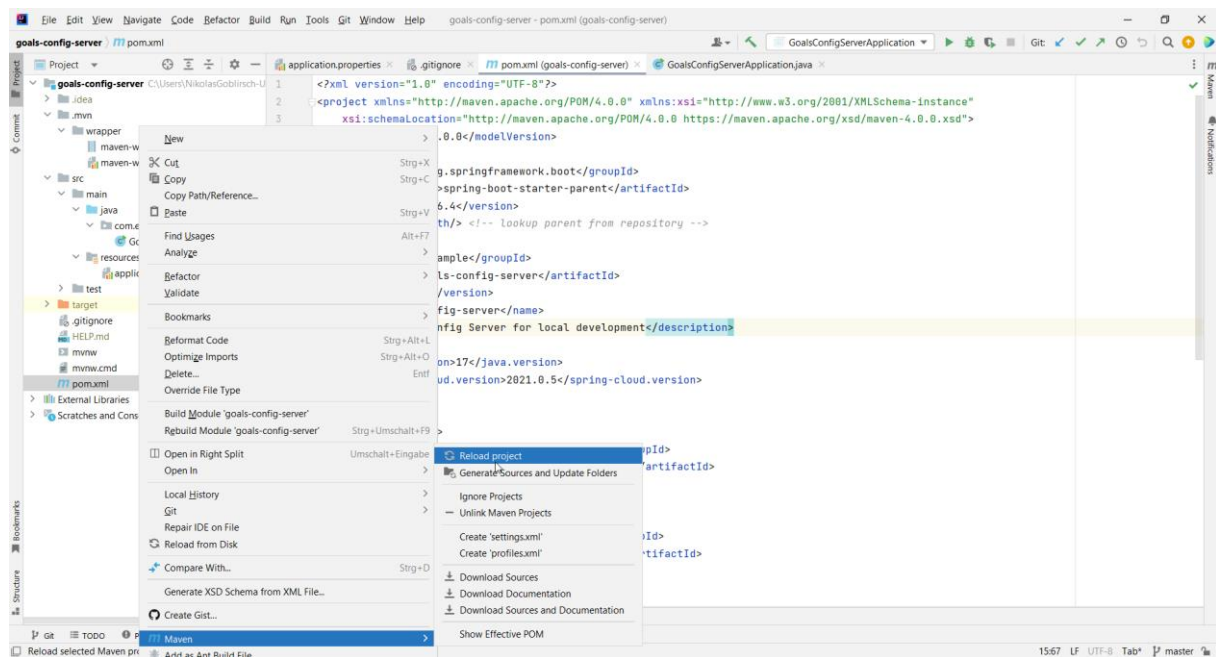
Use your Java IDE to create a new project from version control:



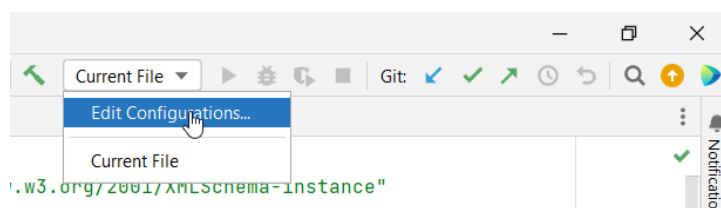
Checkout the repository <https://github.com/NikolasGoblirschUrban/configserver.git> (use GitHub).



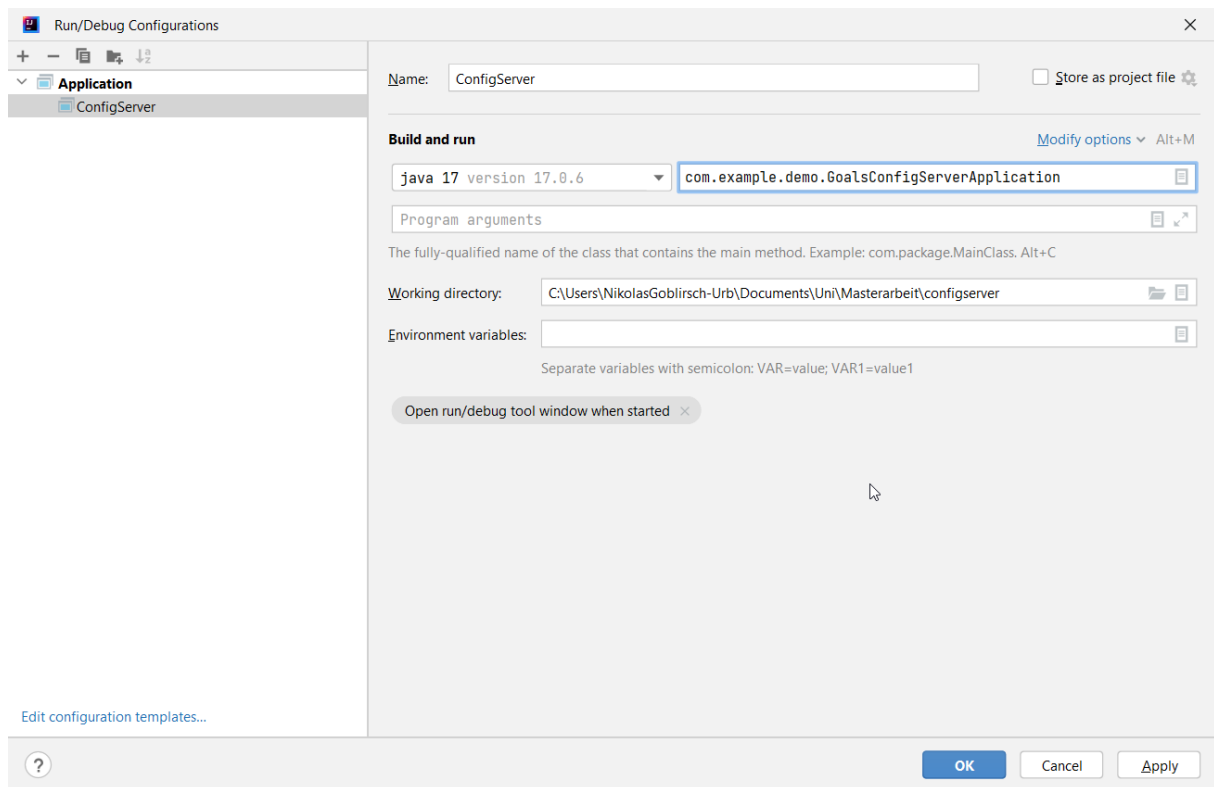
Navigate to the file pom.xml and run maven to fetch the required dependencies.



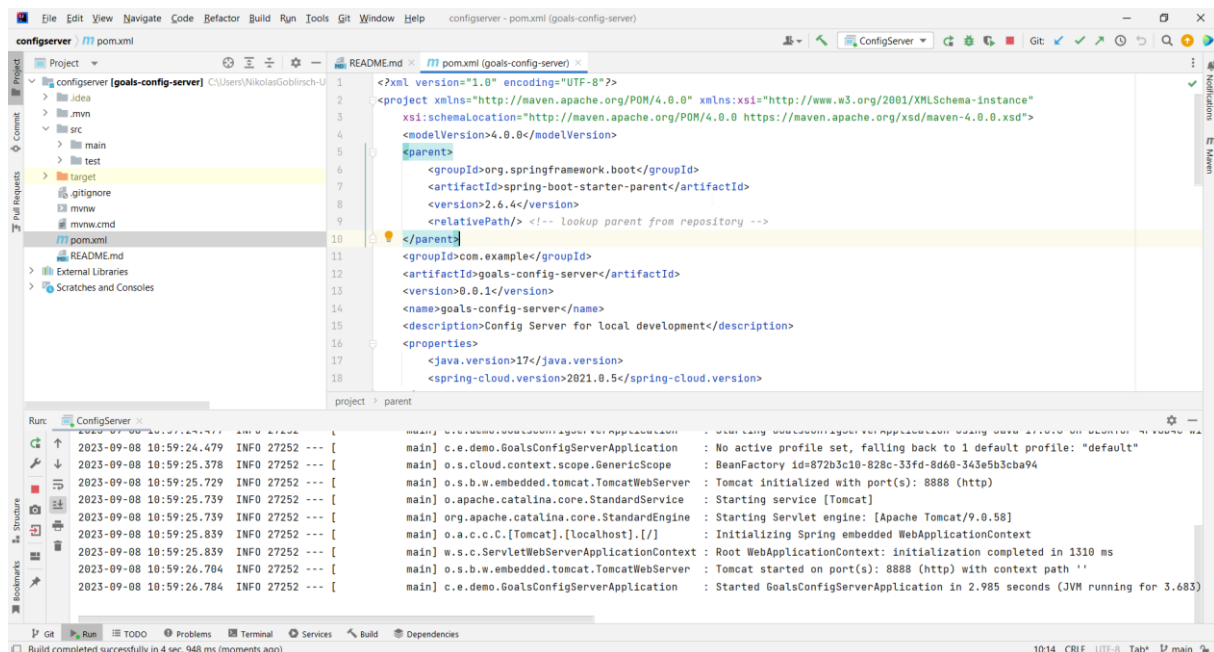
Create a new runtime configuration for the project.



Add a new Application and specify Java 17 and the main class GoalsConfigServerApplication.








You can now run the config server application via the run button.



The application.properties file is already correctly configured and specifies the port the config server runs on and the repository where the config files for the services are stored.

```
server.port = 8888
spring.cloud.config.server.git.uri=https://github.com/NikolasGoblirschUrban/service-configs.git
```

Open the repository. <https://github.com/NikolasGoblirschUrban/service-configs.git>. Here for each service a config file exists. The files are named after the pattern servicename.yml and must have exactly the same name as specified in the application.properties file of the corresponding service the config is for.

 NikolasGoblirschUrban Removed azure stuff and added comments		572920c last week	 6 commits
	goalservice.yml	Removed Azure stuff and added comments	last week
	itemservice.yml	Removed azure stuff and added comments	last week
	userservice.yml	Removed azure stuff and added comments	last week

The only required change is the database connection. Below datasource the url and credentials of the database server used for the service need to be specified. The URL also contains the name of the database with url/database-name. Keep the database names in each file.

After changing the configuration commit the changes and restart the config server application.

2. Microservices

To run the three microservices goalservice, itemservice and userservice make sure that you have configured the files from the config server with the database connection, you have the config server running locally and you have started ActiveMQ as described in the chapter prerequisites.

The required repositories are:

- <https://github.com/NikolasGoblirschUrban/goalservice.git>
- <https://github.com/NikolasGoblirschUrban/itemservice.git>
- <https://github.com/NikolasGoblirschUrban/userservice.git>

Repeat the following steps from chapter Config Server for each repository:

- Checkout new project from version control
- Run Maven to fetch dependencies
- Create new runtime configuration

Before you start the application you need to adapt the application.properties file:

```
#comment next three lines out for server deployment.
spring.application.name=goalservice
spring.config.import=configserver:http://localhost:8888/
eureka.client.enabled=false

#data seeder config: set to true to seed initial data, truncate database tables before
goalservice.dataseeder.enabled=true

#🔥turn syncs with other microservices on and off
goalservice.event-producer.enabled=false
```

Leave the spring.application.name, spring.config.import and eureka.client.enabled unchanged. Set dataseeder.enabled to true and event-producer.enabled to false.

Start the service and wait until the service finishes seeding the data into the database. Then stop the service and set dataseeder.enabled to false and event.producer enabled to true. The service now has data and is ready to be used.

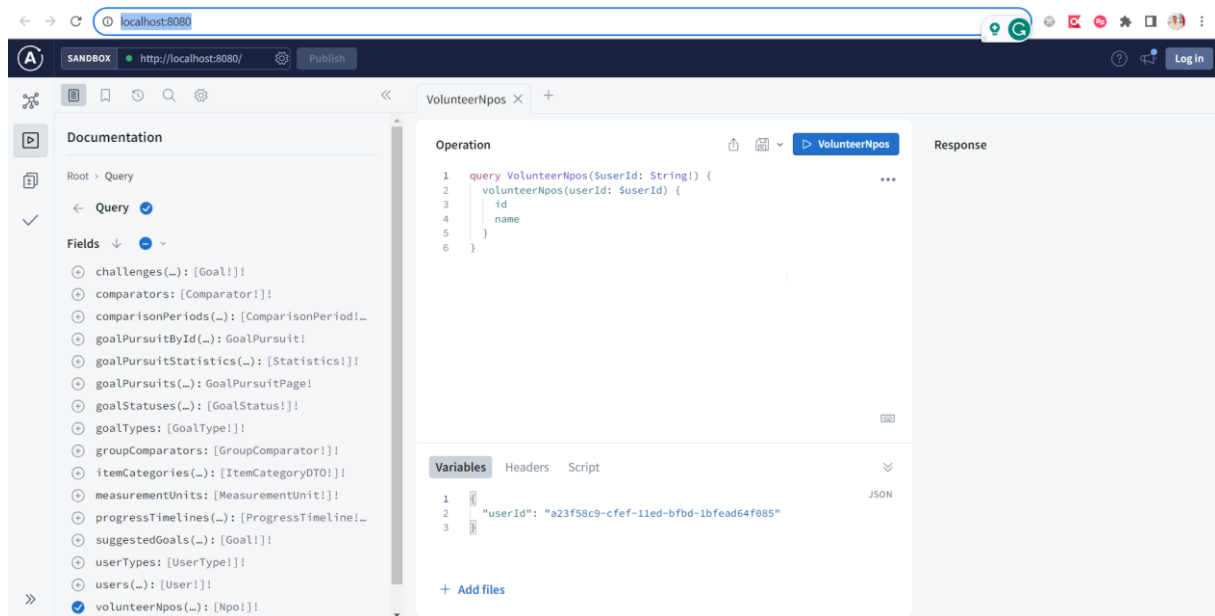
When you want to reseed the data delete the tables from the database manually or truncate the data inside the tables, set dataseeders to true and syncs to false and restart the service.

To adapt the microservices read the documentation on how to write Spring Boot Applications.

3. Goals BFF (Apollo Server)

Before you want to start and use the backend for frontend application make sure that all the backend services are running.

Use an IDE for web development (e.g. WebStorm) Again checkout a new project from version control. Use the terminal inside your ides and navigate to the directory of the project the package.json file is in (main directory, should be the currently active directory). Run the command `npm install` to fetch the required dependencies. Run `npm start` to start the service. Open a browser and navigate to <http://localhost:8080/>.

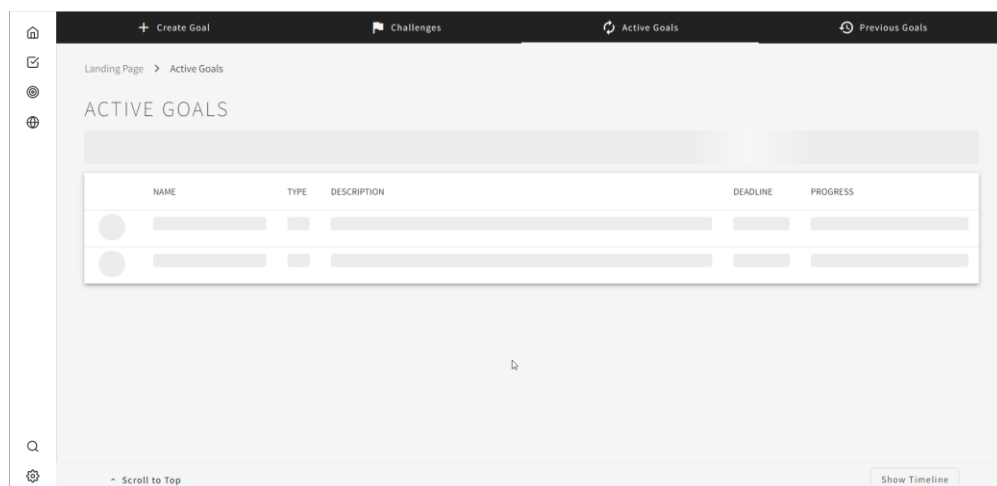


The page shows the GraphQL schema that is defined in the BFF and can be used to click together queries and mutations that are used by the frontend to load data from the backend or save data in the backend.

To adapt the GraphQL Schema read the documentation for `apollo-type-graphql`. To understand how to use `graphql` in the frontend read the `apollo client` documentation.

4. Frontend

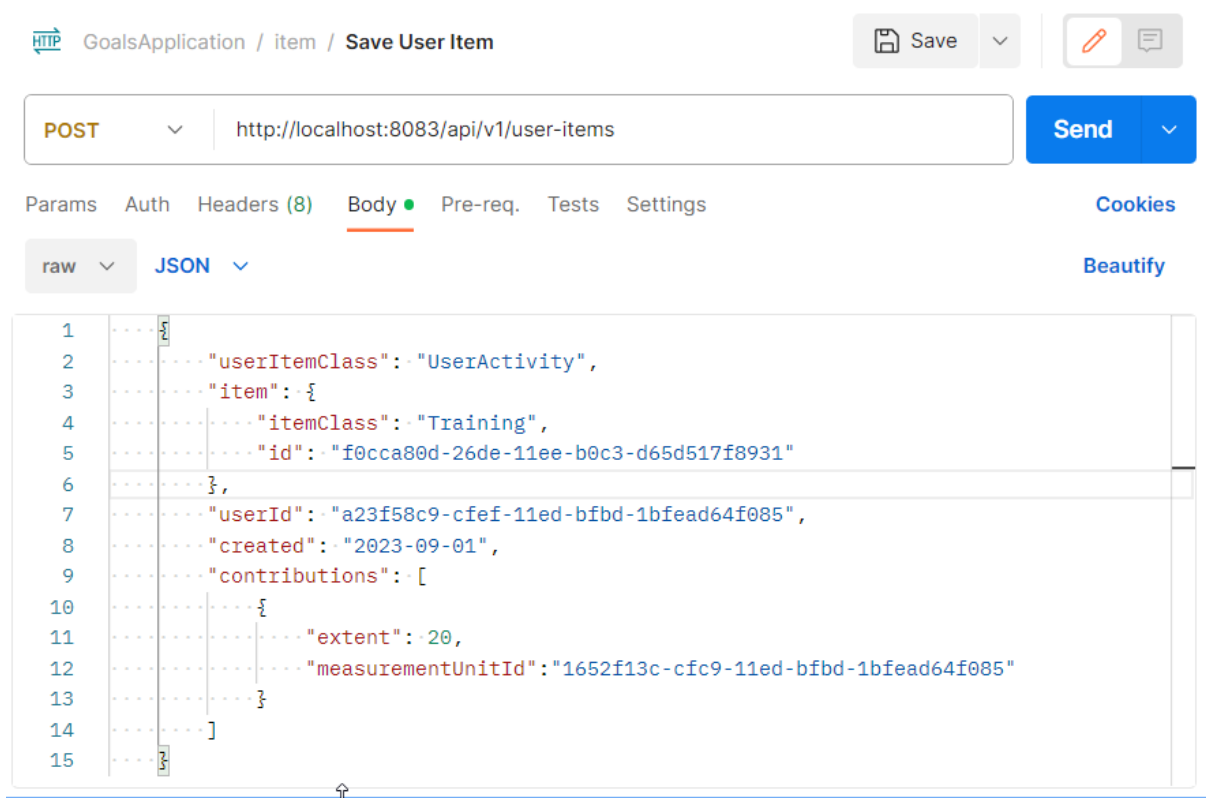
Use an IDE for web development and checkout a new project from version control. Again, run `npm install` to fetch the dependencies. Then run `ng serve` to start the frontend. Navigate to <http://localhost:4200/> and click on the goals menu item.



Make sure that the graphql server is running and all the backend services are running.

5. Change progress of Goal and trigger evaluation

To change the progress of an existing goal open Postman and use the save user items endpoint. All you need to do is specify the correct user id, date (goal only counts if the user item is created after the goal has started, it also only counts for currently active goals) and item category id (category that is used for the goal or a child category). For UserActivites you must specify the goal contributions and for UserAchievements the validFrom date instead.



All required ids can be found in the csv files stored in `resources/data` directory of each service.

Helpful documentation

Name	Link
Angular CLI	https://angular.io/cli
Angular	https://angular.io/docs
Npm	https://www.npmjs.com/
Node.js	https://nodejs.org/de
Spring Cloud Config Server	https://docs.spring.io/spring-cloud-config/docs/current/reference/html/#_quick_start
Spring Boot Applications	https://spring.io/projects/spring-boot
Spring Cloud	https://spring.io/projects/spring-cloud
Apollo Type GraphQL	https://typegraphql.com/docs/introduction.html
Apollo GraphQL Client	https://www.apollographql.com/docs/react/data/queries
ActiveMQ	https://activemq.apache.org/getting-started