Modality Documentation

Table of Contents

Introduction

What is Modality?

Modality is an open-source, Java-based, hospitality-oriented booking system. Multiple organisations, events and properties are supported, together with a wide range of hospitality-related add-ons, such as meals, transport, translation, special needs and more.

Why the name?

'Modality' is a fusion of two words, 'modular' and 'hospitality', a nod to both the software's design and purpose.

Open-source

Modality is licensed under the Apache License Version 2.0, is completely free to use and modify, and is available on GitHub.

The Modality Architecture

Layers

Modality is divided into layers of functionality, shown below:

Layer	Repository	Example modules
Business logic (top layer)	modality	modality-event, modality-hotel, modality-restaurant, modality-catering
Ecommerce	modality	modality-ecommerce
CRM	modality	modality-crm
Base	modality	modality-base
WebFX Stack	webfx-stack	webfx-stack

Layer	Repository	Example modules
WebFX (bottom layer)	webfx	webfx-kit

Business logic

The highest layer of the architecture consists of business-specific modules implementing logic for events, hotels, restaurants etc. This is a customisable layer, and developers can choose to add only the modules they need, as well as provide their own.

Ecommerce

The next layer down is the ecommerce layer. This provides a generic domain model for ecommerce, which models sales, accounts etc. It is the location for payment gateway integration and ecommerce-specific UIs.

CRM

The CRM layer provides the essential CRM features, including customer accounts, integrated mailing system etc.

Base

The Base layer is a fully operational implementation of the webfx-stack layer beneath, based on the Postgres database. This layer is a pure technical solution that isn't bound to any specific domain, and so is large in application scope.

WebFX Stack

The WebFX Stack layer provides an opinionated framework for developing enterprise applications with WebFX. This layer is responsible for communication between client and server (using a WebSocket bus), UI routing, ORM, push notification, auth, i18n, etc. The layer provides interfaces in all cases, but not always full implementations, so that it can be adapted to any kind of system.

Unlike most Java frameworks, this layer works principally on the client side, following the trend initiated by mobile apps where most of the application code has been moved to the client and can run offline.

It is designed to work with JavaFX (for example, i18n provides JavaFX bindings for use with any kind of control; and the authorisation framework automatically enables/disables and shows/hides controls depending on user access).

WebFX

WebFX is the foundation layer, providing a web port of JavaFX (in the webfx-kit module) that can be compiled by GWT together with your application code. It is a Java-based cross-platform solution that can be used in any domain.

All-Layer Aggregation

Modality ships with the `modality-all` module, which aggregates together the full set of modules across all layers, for use by developers right away.

Technologies

Modality is developed using the following technologies:

Technology	Purpose	Version
Java	Codebase	18
JavaFX	Desktop + mobile user interfaces	18
WebFX	Web user interfaces	Latest (Beta)

…consists of three end-user client applications:

Application	Used By	Compilation Toolchain
Front-Office (Mobile)	Customer	Gluon
Front-Office (Web)	Customer	WebFX + GWT
Back-Office (Web)	Administrator	WebFX + GWT

…two developer client applications:

Application	Used By	Compilation Toolchain
Back-Office (Desktop)	Developer	JavaFX
Front-Office (Desktop)	Developer	JavaFX

…one web server:

Application	Purpose	Version
Vert.x	Interface between client apps and back-end services; serves the SPA	Latest

…and depends on the following services:

Service	Purpose	Version
Postgres	Database	14.2
Redis	Session management	6.2.6
Flyway	Database schema updates	Latest

The services are orchestrated by Docker when running Modality on development machines

(instructions given later in this document).

Native applications

Web apps

Modality is the first large-scale Java project to use WebFX - a toolkit that transpiles JavaFX applications into pure JavaScript web apps for direct execution in the browser.

Mobile apps

Modality uses the Gluon toolchain to compile the codebase into native, installable apps ready for inclusion into the Google Play and Apple App stores.

Desktop apps

Modality also provides desktop apps, which provide exactly the same UI as the web apps generated from the same source. This is useful for developers, allowing Java code to be rapidly developed and tested via the desktop, before subsequent transpilation into JavaScript and mobile (which takes time).

Installation

1. Install Java

Modality is developed entirely in the Java language, and requires at least JDK 17+. Check whether this is installed:

```
java --version
```

If it is not installed, or is an older version, please refer to this guide.

2. Install IntelliJ IDEA

We develop Modality using the free, community edition of IntelliJ IDEA, and recommend you install this if you do not already have an IDE. IntelliJ allows you to easily compile and run the Modality server and clients, for the purpose of local development and testing.

All subsequent IDE-based examples given in this documentation will be based on IntelliJ.

3. Install Docker

During development, Modality uses Docker for all external services, including the database and the in-memory datastore for sessions.

Please install Docker on your local machine if you do not have it already. If using a Mac, the easiest way is to install using `brew`. Please provide Docker with a minimum of 8GB of RAM, ideally more.

Insufficient RAM may result in `java.lang.OutOfMemoryError` errors when importing the modality-dev-db.

4. Install Git

A git client is needed to retrieve the Modality codebase from GitHub. Check if git is installed:

```
git --version
```

If it is not installed, you may wish to refer to this guide.

5. Create the Modality root

```
mkdir -vp modality
export MODALITY_ROOT=${PWD}/modality
```

6. Clone the codebase

Git clone the Modality codebase via the terminal (or IntelliJ etc):

```
cd $MODALITY_ROOT
git clone https://github.com/mongoose-project/modality.git .
```

7. Prepare Docker environment variables

Environment variables store the Postgres database name, username and password. Defaults are provided in the `.env-template`. Use this template file as the basis for your Docker-based configuration, by creating an `.env` file from it. You may leave the defaults, or provide new values accordingly:

```
cd $MODALITY_ROOT/docker
cp .env-template .env
source .env # make the environment variables available to the shell
```

8. Build the Docker containers

```
cd $MODALITY_ROOT/docker
docker-compose build --no-cache
```

9. Start the containers to build the database

```
cd $MODALITY_ROOT/docker
docker-compose up
```

The database scripts are stored in the `modality-base/modality-base-server-datasource/src/main/resources/db/` folder, and are executed sequentially by the Flyway database version control container.

Please allow several minutes for Flyway to complete. Once finished, you will now up and running with all the external services that Modality depends on.

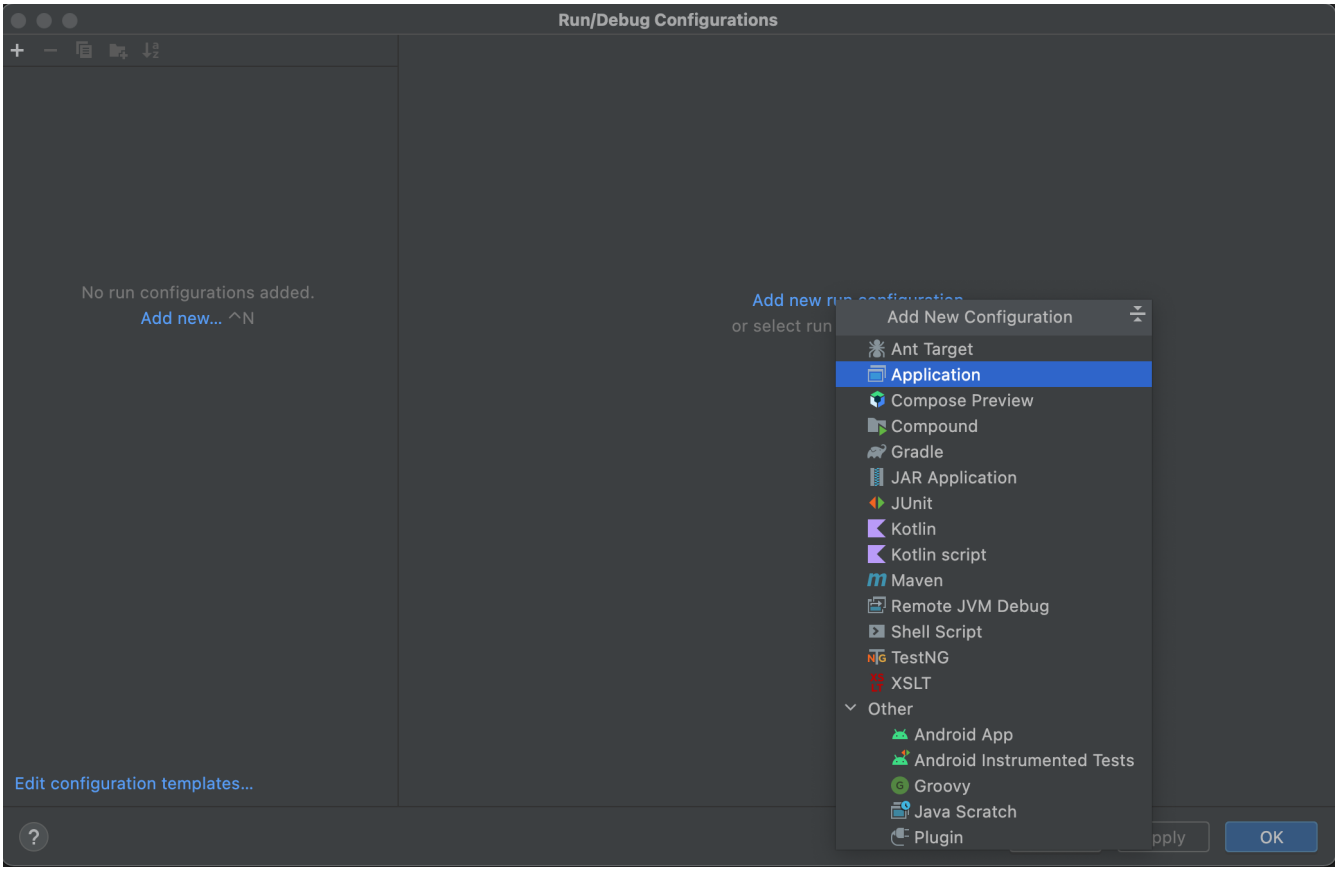
Configure Modality for Development

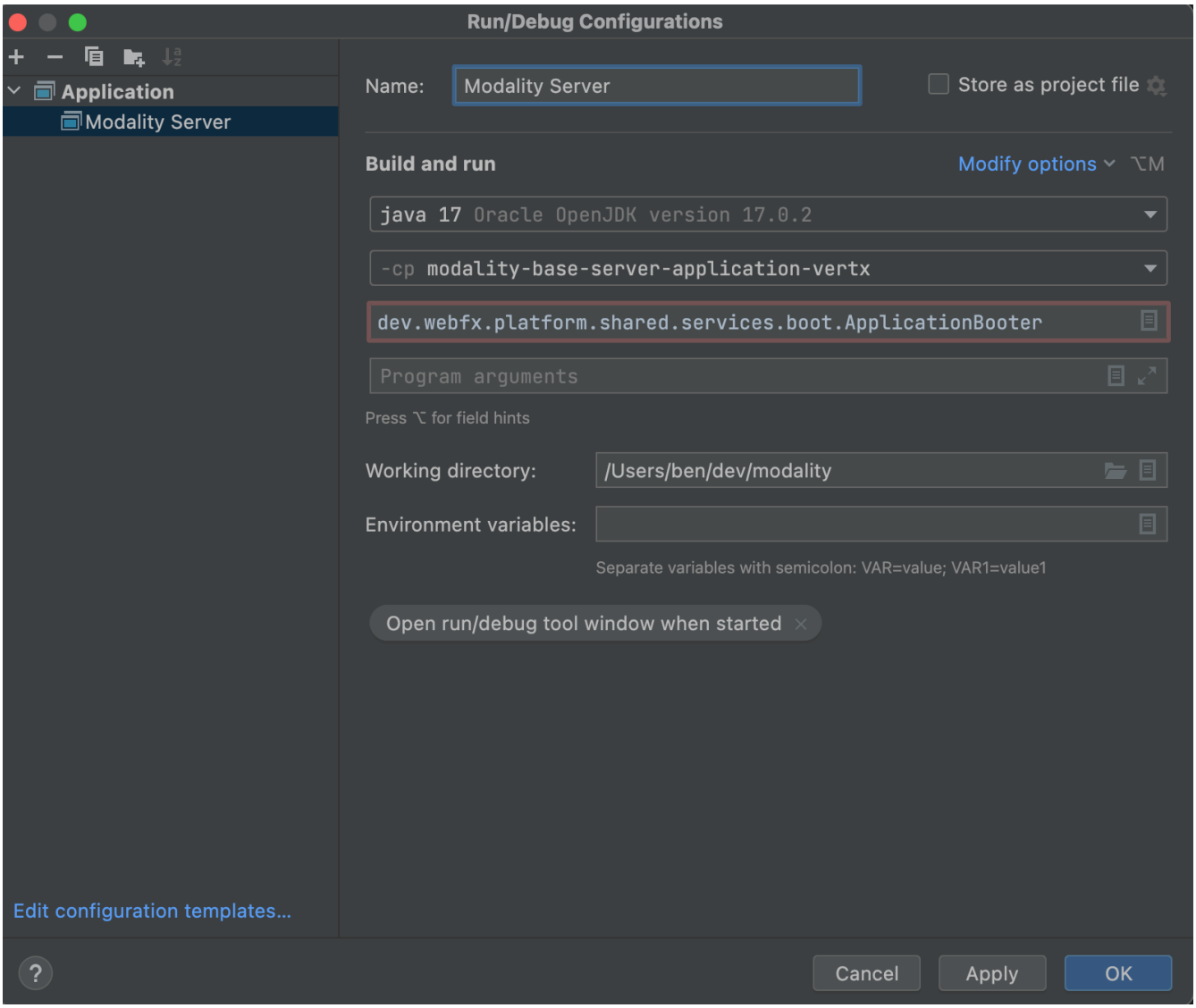
Create a Run Configuration for Modality Server

In order to run any of the Modality client applications, the Modality Server should first be running. The Modality Server is a Vert.x server that proxies requests to the database and is responsible for establishing and maintaining user sessions.

The easiest way to stand up the server locally is to create an application run configuration in your IDE.

In the IntelliJ menu, click `Run → Edit Configurations` to display the following dialog, and populate with the same details:


Run/Debug Configurations
No run configurations added.
Add new... ^N
Add new run configuration...
or select run
Add New Configuration
Ant Target
Application
Compose Preview
Compound
Gradle
JAR Application
JUnit
Kotlin
Kotlin script
Maven
Remote JVM Debug
Shell Script
TestNG
XSLT
Other
Android App
Android Instrumented Tests
Groovy
Java Scratch
Plugin
Edit configuration templates...
OK

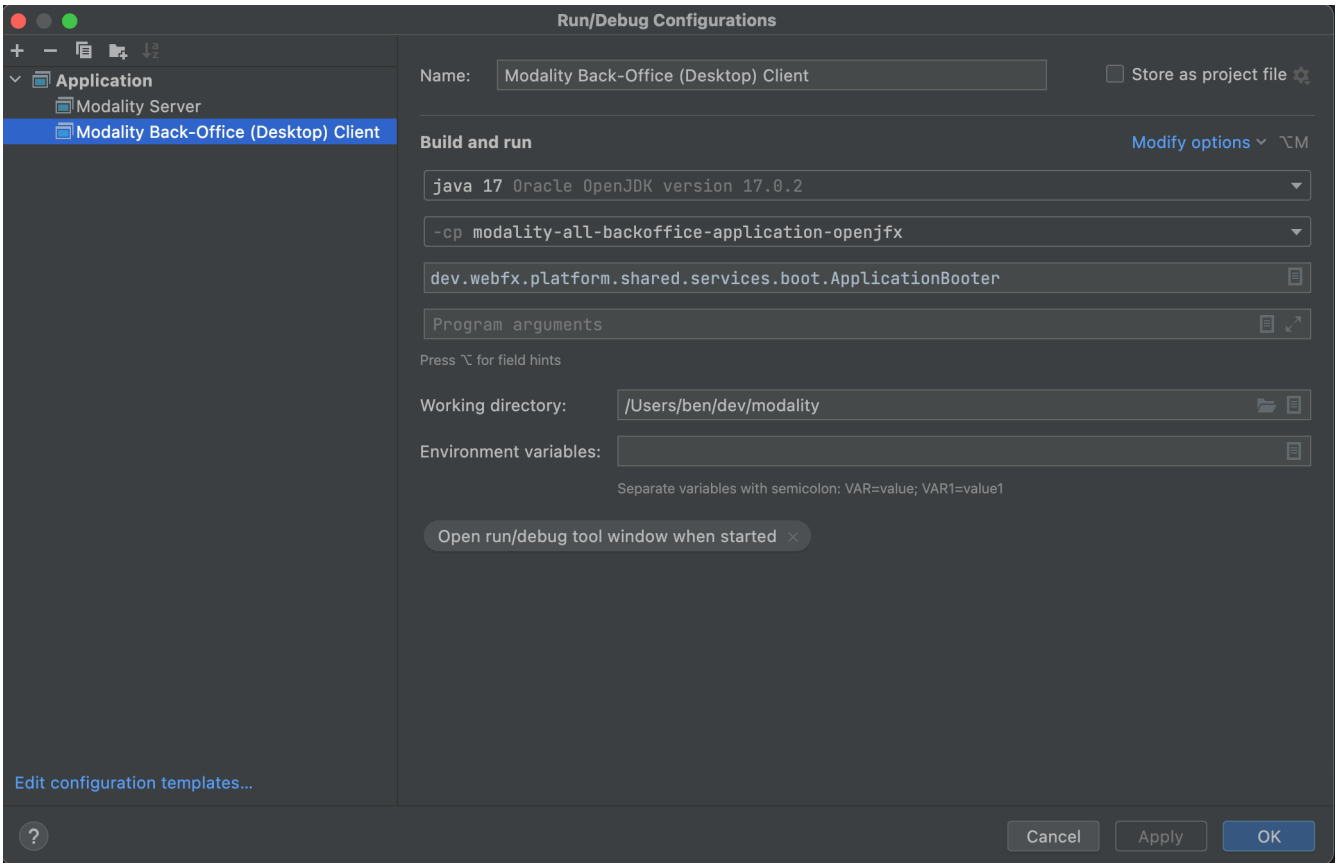

Run/Debug Configurations
Application
Modality Server
Name:
Modality Server
Store as project file
Build and run
Modify options
⌥M
java 17 Oracle OpenJDK version 17.0.2
-cp modality-base-server-application-vertx
dev.webfx.platform.shared.services.boot.ApplicationBooter
Program arguments
Press ⌥ for field hints
Working directory:
/Users/ben/dev/modality
Environment variables:
Separate variables with semicolon: VAR=value; VAR1=value1
Open run/debug tool window when started
Edit configuration templates...
Cancel
Apply
OK

Click 'OK' to save the configuration and close the dialog.

Create a Run Configuration for Modality Back-Office (Desktop) client

The Modality Back-Office (Desktop) client is an application used by developers of Modality, and emulates the web user interface used by administrators of the system.

Create another run configuration and populate it with the details given in the screenshot below:



Click 'OK' to save the configuration and close the dialog.

Create a Run Configuration for Modality Back-Office (Web) client

Documentation not yet available.

Create a Run Configuration for Modality Front-Office (Mobile) client

Documentation not yet available.

Create a Run Configuration for Modality Front-Office (Web) client

Documentation not yet available.

Run Modality on Development

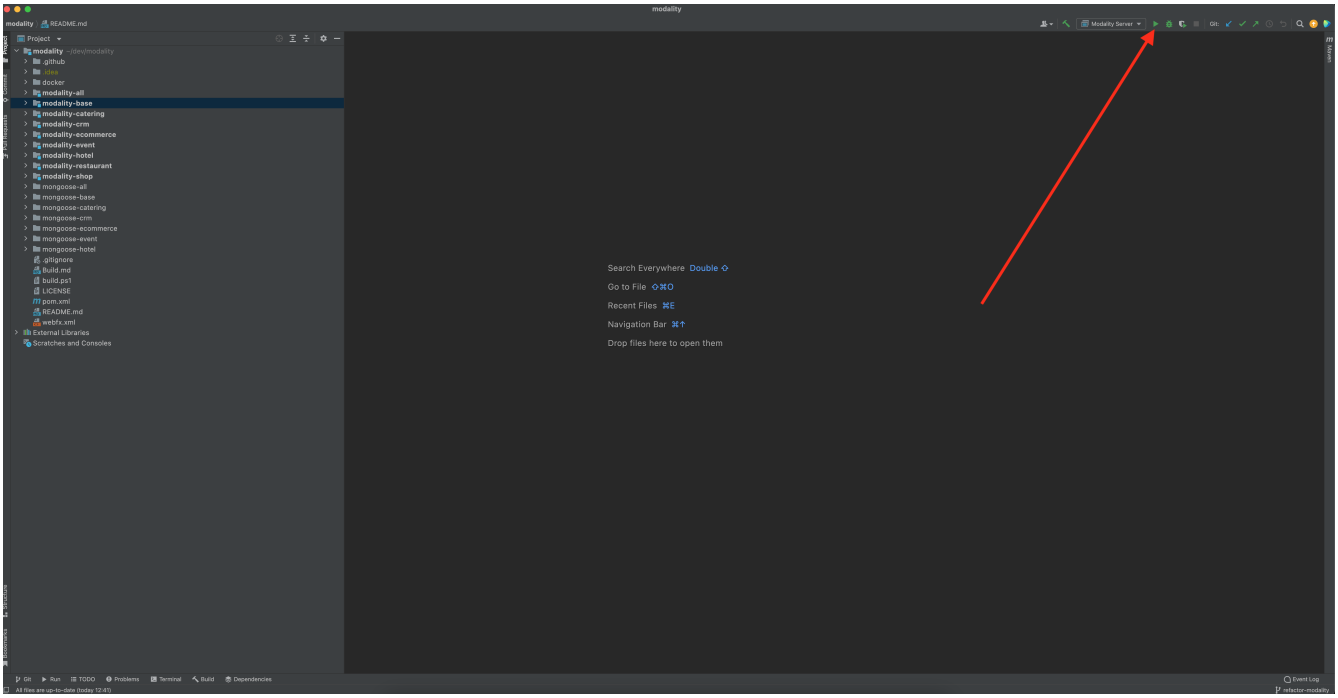
The Modality clients run independently of each other, but all require the Modality Server to be running, which in turn requires Docker to be running the service containers described above. Therefore, the first two steps below are mandatory before running one or more of the Modality clients locally.

Ensure Docker is running *

```
cd $MODALITY_ROOT/docker
docker-compose up
```

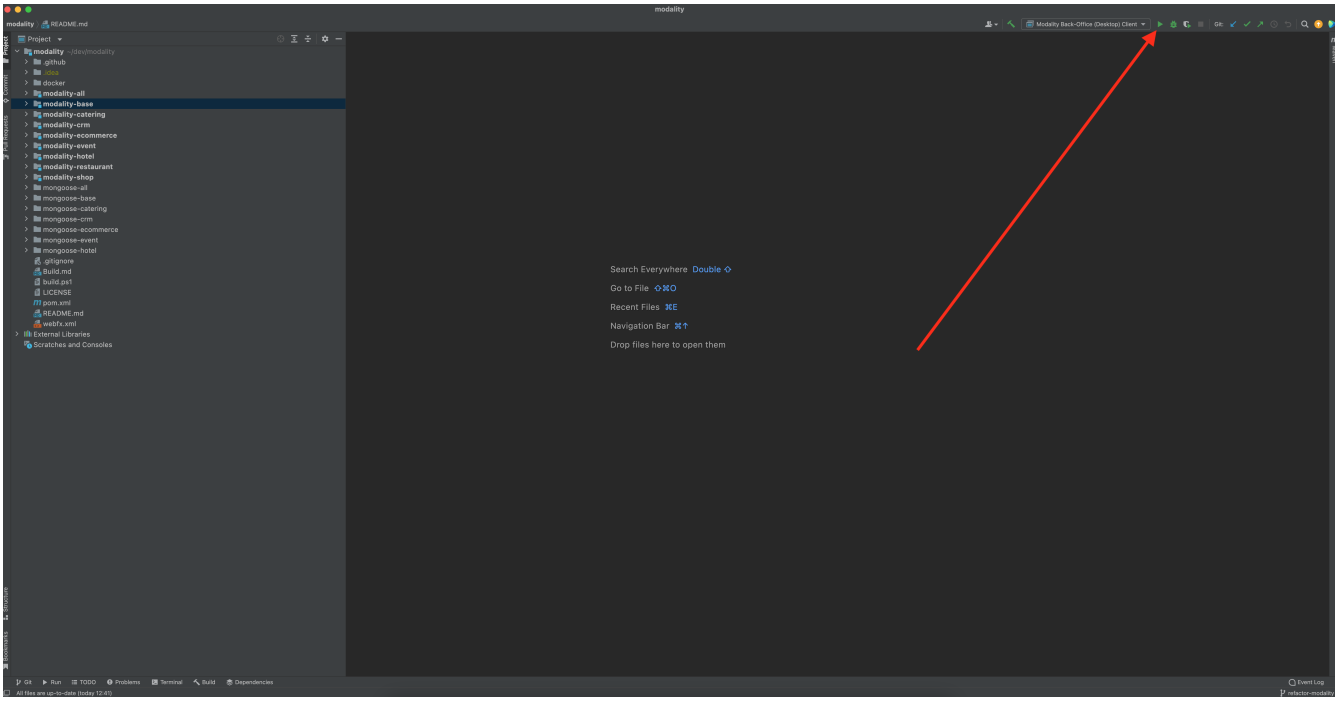
Start the Modality Server *

Start the Modality server by executing the run configuration:

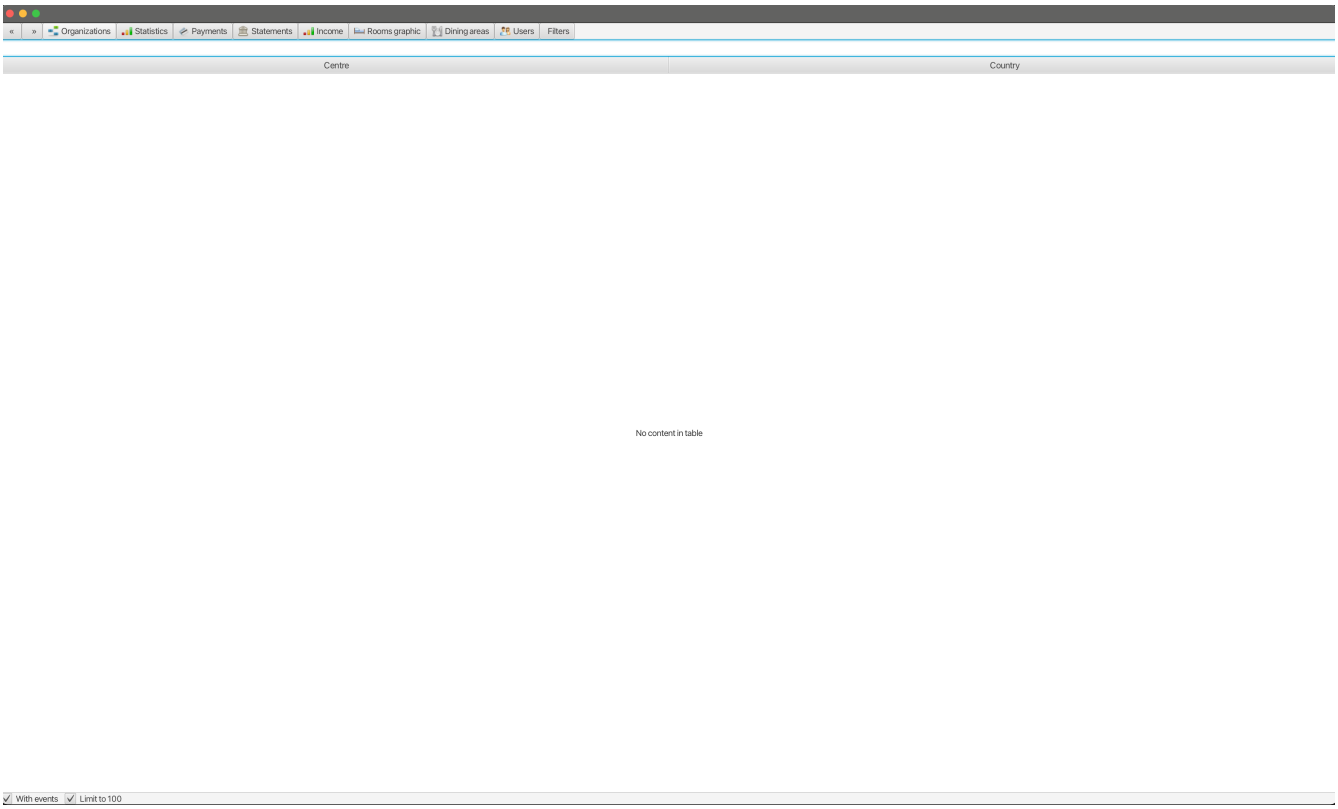


Start Modality Back-Office (Desktop) client

Start the Modality Back-Office (Desktop) client by executing run configuration:



The Back-Office (Desktop) client should then display:



Start Modality Back-Office (Web) client

Documentation not yet available.

Start Modality Front-Office (Mobile) client

Documentation not yet available.

Start the Modality Front-Office (Web) client

Documentation not yet available.

Modality Database

All database setup scripts are stored in the `modality-base/modality-base-server-datasource/src/main/resources/db/` folder, and are numbered in order of execution. Execution of the database scripts is performed automatically by the Flyway container, which runs on Docker startup. All the data is stored on the host, in directory:

```
$MODALITY_ROOT/docker/data/postgres/*
```

This provides persistence, and the container can be safely shut down and restarted without losing data.

Any new database scripts must be:

① added to the same `modality-base/modality-base-server-datasource/src/main/resources/db/` folder

② named according to the convention used in the folder: `V{number}__{desc}.sql`

Once a new script has been added to the folder, the Flyway container should be restarted, in order to apply the change. The easiest way to do this is to simply restart docker-compose:

```
cd $MODALITY_ROOT/docker
docker-compose down
docker-compose up
```

Modality Development Database

The Modality project additionally provides a development database that is pre-populated with test data, available from the modality-dev-db repository.

If you wish to import this database, you will need to:

① shut down the Modality server

② shut down the docker containers

③ delete the `docker/data/` folder

④ download the modality-dev-db repository

⑤ decompress the `V0001__modality_dev_db.sql.zip` file in the modality-dev-db repository

⑥ move the unzipped `V0001__modality_dev_db.sql` to the `modality-base/modality-base-server-datasource/src/main/resources/db/` folder

⑦ move all the other scripts temporarily out of the folder

⑧ restart the docker containers - this will auto-import the development database

⑨ wait until the import is complete. Due to the size of the development database, it can take 20+ minutes to import. Modality will not be usable during this time.

Modality Session

The session data is controlled by the docker-based Redis container and is not persisted locally. The data persists only as long as the container is running.

Using Docker

Connect to the Docker database container

Connection is easily made via any Postgres client (e.g. DBeaver). Use the following credentials (contained within the `docker/.env-template` file):

- Server: 127.0.0.1
- Port: 5432
- Database: modality
- User: modality
- Password: modality

Connect to the Docker session container

Connection can be made through the Docker terminal:

```
cd $MODALITY_ROOT/docker
docker exec -ti session /bin/sh
redis-cli
keys *
```

Shut down Docker

```
cd $MODALITY_ROOT/docker
docker-compose down
```

Destroy & rebuild the Docker containers

Sometimes you will want a fresh set of containers. The simplest way to do this is:

```
cd $MODALITY_ROOT/docker
docker-compose down
docker ps -a # Lists all Docker containers
docker rm <container-id> # Remove any docker containers listed
docker images # Lists all Docker images
docker image rm <image-id> # Remove any docker images listed
docker volume ls # Lists all Docker volumes
docker volume rm <volume-id> # Remove all docker volumes listed
docker system prune # Removes build cache, networks and dangling images
rm -rf data # Removes locally stored database tables
```

You can now rebuild the Docker containers:

```
docker-compose build --no-cache
docker-compose up
```

Deploy Modality to Heroku

Procedures for this coming soon!