SUPERSEDE

Developer manual

# Prerequisites

* Gradle 2.10 ( <http://gradle.org/gradle-download/> )
* PostgreSQL 9.4 ( <http://www.postgresql.org/download/> )
* Git (<https://git-scm.com/downloads> )
* Redis 2.8 (<http://redis.io/download> or <https://github.com/MSOpenTech/redis/releases/tag/win-2.8.2104> for windows)
* Apache httpd 2.4 (<http://www.apachelounge.com/download/> for windows)
* Spring Tool Suite (https://spring.io/tools/sts/all)

# Set up workspace

1. Clone supersede frontend git repository (<https://github.com/supersede-project/frontend>)
2. Create one or more Postgres databases with the same owner
3. Run following scripts to configure the databases:

<supersede\_dir>/conf/postgreSql/schema/restore\_schema.sh $user $password $DB1 $DB2 $DBn

<supersede\_dir>/conf/postgreSql/schema/ apply\_updates.sh $user $password $DB1 $DB2 $DBn

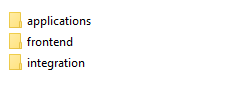
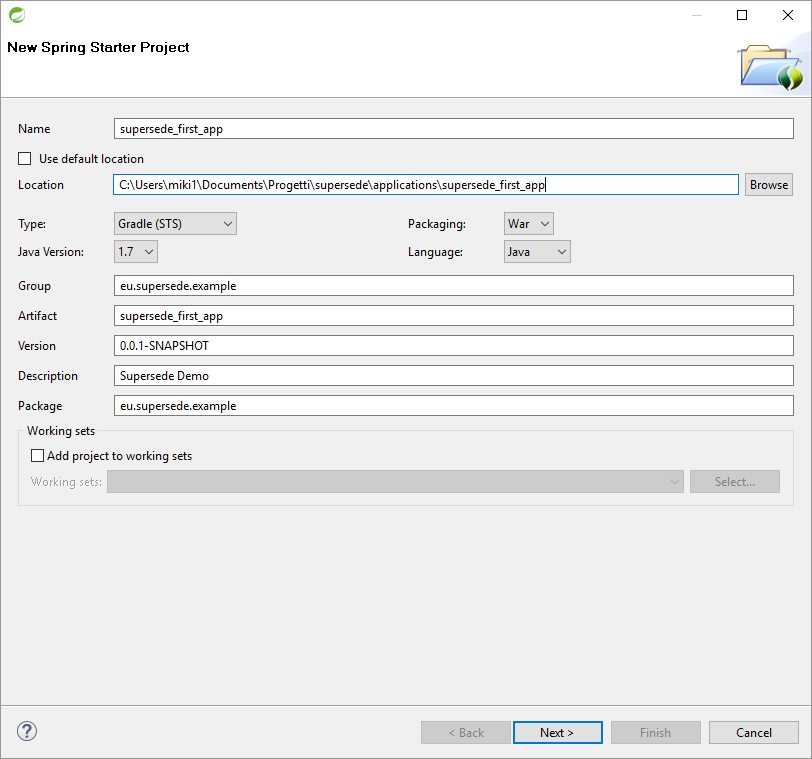
1. Build all the projects:

cd <supersede\_dir>/applications/

./build\_all.sh

1. In STS, import all the gradle projects from <supersede\_dir>/applications
2. Configure apache httpd server to reverse proxy to microservers, a basic configuration file is located in <supersede\_dir>/conf/httpd/httpd.conf
3. Configure <supersede\_dir>/multitenancy.properties and if.properties based on your databases and integration framework configurations.
4. In STS, run supersede-frontend and admin-user-manager-app projects
5. Open your browser and go to “localhost”

# Make your first application

1. Create “applications” folder at same depth of “frontend” and “integration” folders.  
   
2. In STS, create a new Spring Starter Project.  
   
3. Open build.gradle file and add the following required dependencies:

compile('org.springframework.boot:spring-boot-starter-security')

compile("org.springframework.boot:spring-boot-starter-jdbc")

compile("org.springframework.boot:spring-boot-starter-data-jpa")

compile("org.postgresql:postgresql:9.4.1208.jre7")

compile("org.springframework.boot:spring-boot-starter-mail")

compile("org.springframework.session:spring-session:1.2.0.RELEASE")

compile("org.springframework.boot:spring-boot-starter-redis")

compile files('../../frontend/applications/supersede-client/build/libs/supersede-client-0.0.1-SNAPSHOT.jar')

//required by integration

compile files('../../integration/IF/API/eu.supersede.if.api/build/libs/eu.supersede.if.api.jar')

compile ("org.apache.axis2:axis2:1.6.1")

compile ("org.apache.axis2:axis2-transport-http:1.6.1")

compile ("org.apache.axis2:axis2-transport-local:1.6.1")

compile files('../../integration/IF/API/eu.supersede.if.api.test/lib/org.wso2.carbon.um.ws.api.stub\_4.2.2.jar')

compile files('../../integration/IF/API/eu.supersede.if.api.test/lib/org.wso2.carbon.um.ws.api\_4.2.2.jar')

compile files('../../integration/IF/API/eu.supersede.if.api.test/lib/org.wso2.carbon.user.api\_4.2.0.jar')

compile files('../../integration/IF/API/eu.supersede.if.api.test/lib/org.wso2.carbon.user.core\_4.2.0.jar')

compile files('../../integration/IF/API/eu.supersede.if.api.test/lib/org.wso2.carbon.utils\_4.2.0.jar')

compile ("ca.juliusdavies:not-yet-commons-ssl:0.3.9")

1. Delete “.gradle” and “.setting” folders and “.classpath” and “.project” files.
2. Run “gradle eclipse” from command line to update the project dependencies.
3. Open application.properties file and add following lines:

server.port=8083

logging.level.eu.supersede.example=DEBUG

spring.jpa.database=POSTGRESQL

spring.jpa.show-sql=false

spring.jpa.hibernate.ddl-auto=none

spring.redis.host=localhost

spring.redis.port=6379

security.sessions=NEVER

1. Create in src/main/resources the file “wp5\_application.properties” and add following lines:

application.name=supersede\_first\_app

application.unsecured.urls=/open\_apis/\*\*

application.multitenancy.models.packages=eu.supersede.example.model

application.label=Supersede first app

application.label.it=Supersede prima applicazione

application.pages=example

application.home=example

application.page.example.profiles=ADMIN

application.page.example.label=Example

application.page.create\_user.label.it=Esempio

application.gadgets=example\_gadget

application.gadgets.example\_gadget.profiles=ADMIN

1. Delete “ServletInitializer.java” file.
2. Replace “SupersedeFirstAppApplication.java” with following lines:

package eu.supersede.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.context.web.SpringBootServletInitializer;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;

import eu.supersede.fe.configuration.ApplicationConfiguration;

@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)

@ComponentScan(basePackages = {"eu.supersede.example", "eu.supersede.fe"})

@EnableGlobalMethodSecurity( securedEnabled = true, prePostEnabled = true )

@EnableJpaRepositories(basePackages={"eu.supersede.example.jpa"})

@EnableRedisHttpSession

public class SupersedeFirstAppApplication extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

ApplicationConfiguration.init();

return application.sources(SupersedeFirstAppApplication.class);

}

public static void main(String[] args) {

ApplicationConfiguration.init();

SpringApplication.run(SupersedeFirstAppApplication.class, args);

}

}

1. Create “eu.supersede.example.model” package, here you can put all the model classes based on your database schema.
2. Create User class in “eu.supersede.example.model”, then add following lines:

package eu.supersede.example.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.Transient;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import com.fasterxml.jackson.annotation.JsonProperty;

@Entity

@Table(name = "users")

@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long userId;

private String username;

private String firstName;

private String lastName;

private String email;

@Transient

private String password;

public User() {

}

public Long getUserId() {

return userId;

}

public void setUserId(Long userId) {

this.userId = userId;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

1. Create “eu.supersede.example.jpa” package, here you can put all the jpa classes based on your database schema.
2. Create UsersJpa interface in “eu.supersede.example.jpa”, add following lines:

package eu.supersede.example.jpa;

import org.springframework.data.jpa.repository.JpaRepository;

import eu.supersede.example.model.User;

public interface UsersJpa extends JpaRepository<User, Long> {

}

1. Create “eu.supersede.example.rest” package, here you can put all the rest classes used to map REST services.
2. Create UserRest class in “eu.supersede.example.rest”, then add following code:

package eu.supersede.example.rest;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.Authentication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import eu.supersede.example.jpa.UsersJpa;

import eu.supersede.example.model.User;

import eu.supersede.fe.security.DatabaseUser;@RestController

@RequestMapping("/user")

public class UserRest {

@Autowired

UsersJpa users;

@RequestMapping("")

public User getUser(Authentication authentication)

{

DatabaseUser currentUser = (DatabaseUser) authentication.getPrincipal();

User u = users.getOne(currentUser.getUserId());

return u;

}

}

1. Create “example.html” file and add following lines:

<script src="supersede\_first\_app/js/example.js"></script>

<div ng-controller="example">

Hello {{user}}

</div>

1. Create “js/example.js” file and add following lines:

var app = angular.module('w5app');

app.controllerProvider.register('example', function($scope, $http) {

$scope.user = "";

$scope.getUser = function()

{

$http({

url: "supersede\_first\_app/user",

method: 'GET'

}).success(function(data){

$scope.user = data.firstName + " " + data.lastName;

}).error(function(err){

console.log(err);

});

};

$scope.getUser();

});

1. Configure apache httpd, adding a new reverse proxy to your project:

ProxyPass /supersede\_first\_app <http://localhost:8083/>

ProxyPassReverse /supersede\_first\_app http://localhost:8083/

1. Restart apache httpd
2. At the same depth of “supersede\_first\_app” folder create a “conf” folder and copy inside multitenancy.properties and if.properties, then configure them.
3. In STS, run “supersede\_first\_app” application and “supersede-frontend”

# How Access Integration Data

In any String @Component, @Repository, @Service and @Controller classes you can have access to ProxyWrapper utility class defining an “autowired” global variable “ProxyWrapper proxy” like in the example below:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.Authentication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import eu.supersede.fe.exception.NotFoundException;

import eu.supersede.fe.integration.ProxyWrapper;

import eu.supersede.fe.security.DatabaseUser;

import eu.supersede.integration.api.datastore.fe.types.User;

@RestController

@RequestMapping("/user")

public class UserRest {

@Autowired

private ProxyWrapper proxy;

@RequestMapping("/current")

public String getUser(Authentication authentication)

{

DatabaseUser currentUser = (DatabaseUser) authentication.getPrincipal();

Long userId = currentUser.getUserId();

User proxyUser = proxy.getFEDataStoreProxy().getUser(currentUser.getTenantId(), userId.intValue(), true, currentUser.getToken());

if(proxyUser == null)

{

throw new NotFoundException();

}

return proxyUser.getFirst\_name() + " " + proxyUser.getLast\_name();

}

}

# How Access Database With Logged User

You can @Autowired your Jpa interfaces to read and write database data when you receive a request from a logged-in user, for example in a REST request:

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import eu.supersede.fe.jpa.ProfilesJpa;

import eu.supersede.fe.jpa.UsersJpa;

import eu.supersede.fe.model.Profile;

import eu.supersede.fe.model.User;

@RestController

@RequestMapping("/user")

public class UserRest {

@Autowired

private UsersJpa users;

@Autowired

private ProfilesJpa profiles;

@RequestMapping(value = "", method = RequestMethod.POST)

public ResponseEntity<?> createUser(@RequestBody User user) {

// re-attach detached profiles

List<Profile> ps = user.getProfiles();

for (int i = 0; i < ps.size(); i++) {

ps.set(i, profiles.findOne(ps.get(i).getProfileId()));

}

user = users.save(user);

HttpHeaders httpHeaders = new HttpHeaders();

httpHeaders.setLocation(ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")

.buildAndExpand(user.getUserId()).toUri());

return new ResponseEntity<>(null, httpHeaders, HttpStatus.CREATED);

}

}

# How Access Database Without Logged User

You can work with database data without a user request using an @Autowired MultiJpaProvider, for example in a @Scheduled function:

@Scheduled(fixedRateString = "${notifier.mail.sender.checkRate}")

public void checkNotifications()

{

Date now = new Date();

Date limit = new Date(now.getTime() - SENDER\_DELAY);

Map<String, NotificationsJpa> notificationsJpa = multiJpaProvider.getRepositories(NotificationsJpa.class);

for(NotificationsJpa nJpa : notificationsJpa.values())

{

//get all notifications not read and not sent via email and created before

List<Notification> ns = nJpa.findByReadAndEmailSentAndCreationTimeLessThan(false, false, limit);

for(Notification n : ns)

{

sendEmail(n);

n.setEmailSent(true);

nJpa.save(n);

}

}

}

# Send Emails

# Create Notifications

# Publish Pages To Navigation Bar

# Publish Gadgets