Promise – start-up guide

This document describes steps needed to take to start developing, testing and using Promise and STS on your computer.

Promise itself contains two separate parts: Front-end and Back-end. While front-end requires different server setup, back-end and STS require almost the same setup. Where different, a note is explicitly stated.

Chapter 1
Prerequisites

Before you start using and developing Promise or STS, you must make sure to have installed and correctly set up some required software.

1.1 Production deployment

In this section we suppose you already have built the project and only want to deploy and run it.

To host the Promise Back-end and/or STS you will need:

1. Database server – PostgresSQL 9.6 recommended but not required
2. Java Runtime Environment 8 (JRE8)
3. Java application server – Apache Tomcat 8 recommended but not required

To host Promise Front-end you will need:

1. Web server – Apache HTTP server 2.4 recommended

1.2 Local development

If you also want to develop and build Promise Back-end and STS, in addition to requirements from 1.1 you will also need:

1. Kotlin compiler
2. Java Development Kit 8 (JDK8)
3. Maven 3

NOTE: It is highly recommended to use IntelliJ IDEA as your IDE since it is already shipped with all of the 3 requirements above.

To build Front-end Apache HTTP Server is not used. Instead you will need:

1. Node Package Manager (NPM 5.6.0 recommended)

Chapter 2

Configuration

For both production deployment and local development Promise Back-end and STS want us to specify a JNDI resource for database connections. Back-end is looking for resource named *jdbc/promise*, STS for *jdbc/sts*.

If you followed previous recommendations and used Apache Tomcat and PostgresSQL, you can specify the resources in file *context.xml* in Tomcat's *conf* directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>

   <!-- ... -->

   <Resource name="jdbc/promise" type="javax.sql.DataSource"
             maxTotal="100" maxIdle="30" maxWaitMillis="10000"
             username="YOUR-USERNAME" password="YOUR-PASSWORD"
             driverClassName="org.postgresql.Driver" auth="Container"
             url="jdbc:postgresql://localhost:5432/promise"/>

   <Resource name="jdbc/sts" type="javax.sql.DataSource"
             maxTotal="100" maxIdle="30" maxWaitMillis="10000"
             username="YOUR-USERNAME" password="YOUR-PASSWORD"
             driverClassName="org.postgresql.Driver" auth="Container"
             url="jdbc:postgresql://localhost:5432/sts"/>
</Context>
```

While the database structure for Back-end is created automatically, for STS you must create tables manually e.g. by following minimal SQL commands:

```
CREATE TABLE IF NOT EXISTS AUTHORITIES (
USERNAME VARCHAR(256),
AUTHORITY VARCHAR(256)
);

CREATE TABLE IF NOT EXISTS USERS (
USERNAME VARCHAR(256),
PASSWORD VARCHAR(256),
ENABLED BOOLEAN
);
```

2.1 Production deployment

Picture 2.1.1 shows the recommended production infrastructure.

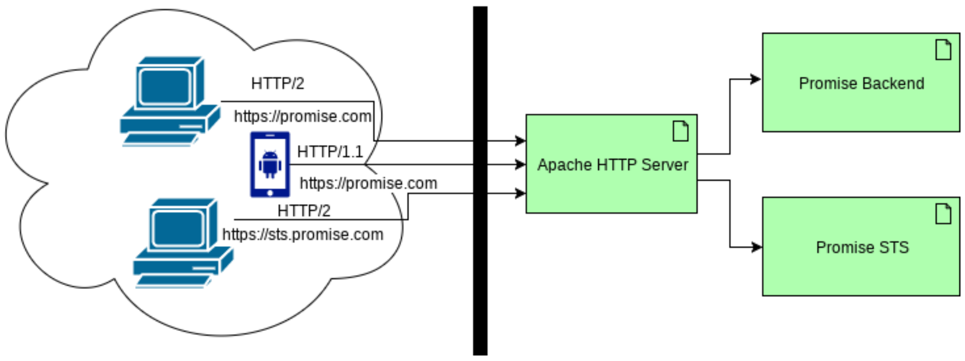

Figure 2.1.1 Recommended production infrastructure

To follow this recommendation you must correctly setup Apache HTTP Server. We suppose you are using Apache Tomcat as an application server with AJP enabled on port 8009 (which is true by default).

We also assume you have enabled SSL, Proxy and HTML Proxy modules in Apache HTTP server and you own a valid certificate for the domain names, therefore TLS and HTTP/2 can be enabled.

2.1.1 Front-end configuration

Production Front-end configuration file *environment.prod.ts* is found in the *src/environments/* directory. You shall adjust URLs in there to your actual values.

2.1.2 Virtual hosts configuration

Virtual host for promise.com can be set up as follows:

```
<IfModule mod_ssl.c>
Protocols h2 http/1.1

<VirtualHost *:443>
   ServerName promise.com
   DirectoryIndex index.html
   DocumentRoot /var/www/promise
   ErrorLog /var/log/httpd/promise.error.log
   CustomLog /var/log/httpd/promise.log combined

   <Directory "/var/www/promise">
      Require all granted
   </Directory>

   <Location /api >
      ProxyPass        ajp://localhost:8009/promise
      ProxyPassReverse https://promise.com
   </Location>

   <Location / >
      RewriteEngine on
      RewriteCond %{REQUEST_URI} !^/api
      RewriteCond %{REQUEST_FILENAME} !-d
      RewriteCond %{REQUEST_FILENAME} !-f
      RewriteRule . /index.html [L]
   </Location>

   SSLCertificateFile       /etc/ssl/promise.com/cert.pem
   SSLCertificateKeyFile    /etc/ssl/promise.com/privkey.pem
   SSLCertificateChainFile  /etc/ssl/promise.com/chain.pem
</VirtualHost>
</IfModule>
```

This causes requests, whose path begins with */api*, to be proxied to Apache Tomcat via AJP. Static files will be served from */var/www/promise* and requests to non-existing files will be responded with *index.html*

Virtual host for sts.promise.com can be set up as follows:

```
<IfModule mod_ssl.c>
Protocols h2 http/1.1

<VirtualHost *:443>
 ServerName sts.promise.com
 ErrorLog /var/log/httpd/sts.error.log
 CustomLog /var/log/httpd/sts.log combined

 #ProxyPass        /manager ajp://localhost:8009/manager/

 ProxyPass                  /     ajp://localhost:8009/sts/
 ProxyPassReverse           /     /sts/
 ProxyPassReverse           /     https://sts.promise.com/sts/
 ProxyPassReverse           /     ajp://localhost:8009/sts/
 ProxyPassReverseCookiePath /sts  /

 ProxyHTMLEnable On
 ProxyHTMLURLMap https://sts.promise.com/sts/ https://sts.promise.com/
 ProxyHTMLURLMap /sts/                        https://sts.promise.com/

 SSLCertificateFile       /etc/ssl/sts.promise.com/cert.pem
 SSLCertificateKeyFile    /etc/ssl/sts.promise.com/privkey.pem
 SSLCertificateChainFile  /etc/ssl/sts.promise.com/chain.pem
</VirtualHost>
</IfModule>
```

This mainly causes hypertext links to be rewritten to a correct URL.

2.2 Local development

Since Promise Back-end depends on slightly modified AFRest. You must build and locally install AFRest artifact from the attached source before building Promise Back-end.

To do so, navigate to the AFRest directory and run:

```
mvn clean install -DskipTests
```

You should also install Front-end NPM dependencies by navigating to the Front-end root directory and running

```
npm install
```

2.3 Creating users

New users for STS must be added manually either using appropriate REST API endpoint (preffered way) or into database. Then the user is automatically registered into Promise during the first time login.

Chapter 3
Local development and testing

If all you wanted was to deploy and start using Promise on production server, you can take a look at the *deploy-*.sh* scripts in Promise root directory to see how to build and deploy artifacts to a production server.

However, for local development some more commands are handy.

3.1 Local development environment

To start a local development environment, follow these steps:

1. Build and start STS
2. Build and start Promise Back-end
3. Build and start Promise Front-end

STS and Back-end can be built and started right in your IDE. To start serving Front-end use:

```
npm run start
```

Front-end changes are then projected immediately, Back-end and STS must be rebuilt and redeployed after each change.

3.2 Testing

To test the application a Mozzila Firefox browser is required for Selenium tests.

You must start Front-end in the test mode at first by running

```
npm run test-env
```

and then you are ready to run the tests using e.g. your IDE.