================================================================================

Cargo Tracker

Applied Domain-Driven Design Blue Prints for Java EE

================================================================================

Overview

========

The project demonstrates how you can develop applications with the Java EE

platform using widely adopted architectural best practices like Domain-Driven

Design (DDD), TDD and agile. The project is directly based on the well known

original Java DDD sample (http://dddsample.sourceforge.net) application

developed by DDD pioneer Eric Evans' company Domain Language and the Swedish

software consulting company Citerus. The cargo example actually comes from

Eric Evans' seminal book on DDD. The original application is written in Spring,

Hibernate and Jetty whereas the application is build entirely on Java EE 7 and

GlassFish 4. It also supports WildFly 8.

The application is an end-to-end system for keeping track of shipping cargo. It

has several interfaces described in the following sections.

For further details on the project, please visit:

http://cargotracker.java.net

Getting Started

===============

The project site has detailed information on how to get started:

https://java.net/projects/cargotracker/pages/Home, especially with NetBeans:

https://java.net/projects/cargotracker/pages/NetBeans.

The simplest steps are the following (no IDE required):

\* Get the project source code: https://java.net/projects/cargotracker/downloads

\* As long as you have Maven set up, navigate to the project source root and

type: mvn package cargo:run

\* To run with WildFly 8x, type: mvn -Pwildfly package cargo:run

\* Go to http://localhost:8080/cargo-tracker

To set up in NetBeans, follow these steps:

\* Set up JDK 7+, NetBeans 7.3+ and GlassFish 4+

(preferably GlassFish 4.1+ and NetBeans 8.0.1+).

\* Open the source code directory in NetBeans - it's just a Maven project,

NetBeans will do the rest for you. As noted in the site instructions on

NetBeans, you may get a few spurious errors due to reported NetBeans bugs.

Just ignore them and proceed with clean/building the application.

\* After the project is built (which will take a while the very first time as

Maven downloads dependencies), simply run it via GlassFish 4.

It's best to use Chrome for the application since it has the best support for

HTML 5 forms, but any HTML 5 capable browser is fine. There are some known

issues due to reported GlassFish 4 bugs - please read the Known Issues section

below.

Exploring the Application

=========================

After the application runs, it will be available at:

http://localhost:8080/cargo-tracker/. Under the hood, the application uses a

number of Java EE (and Java EE 7) features including JSF 2.2, CDI, EJB 3.2,

JPA 2.1, JAX-RS 2, WebSocket, JSON-P, Bean Validation 1.1 and JMS 2.

There are several web interfaces, REST interfaces and a file system scanning

interface. It's probably best to start exploring the interfaces in the rough

order below.

The tracking interface let's you track the status of cargo and is

intended for the general public. Try entering a tracking ID like ABC123 (the

application is pre-populated with some sample data).

The administrative interface is intended for the shipping company that manages

cargo. The landing page of the interface is a dashboard providing an overall

view of registered cargo. The dashboard will update automatically when cargo

is handled (described below). You can book cargo using the booking interface.

One cargo is booked, you can route it. When you initiate a routing request,

the system will determine routes that might work for the cargo. Once you select

a route, the cargo will be ready to process handling events at the port. You can

also change the destination for cargo if needed or track cargo.

The Incident Logging interface is intended for port personnel registering what

happened to cargo. The interface is primarily intended for mobile devices, but

you can use it via a desktop browser. The interface is accessible at:

http://localhost:8080/cargo-tracker/incident-logger/. For convenience, you

could use a mobile emulator instead of an actual mobile device. On Windows,

you can use Microsoft WebMatrix for device emulation. Generally speaking cargo

goes though these events:

\* It's received at the origin port.

\* It's loaded and unloaded onto voyages on it's itinerary.

\* It's claimed at it's destination port.

\* It may go through customs at arbitrary points.

While filling out the event registration form, it's best to have the itinerary

handy. You can access the itinerary for registered cargo via the admin interface.

As you register handling events, the administrative dashboard will be

automatically updated in real time without a page refresh in addition to cargo

state. The cargo handling is done via JMS for scalability and the event

notification to the system happens via the CDI event bus and WebSocket, so you

will see a visible delay of a few seconds after registering the event for the

dashboard to update. While using the incident logger, note that only the load

and unload events require as associated voyage (entering an unnecessary voyage

for other events will result in an error).

You should also explore the file system based bulk event registration interface.

It reads files under /tmp/uploads. The files are just CSV files. A sample CSV

file is available under src/main/resources/handling\_events.csv. Sucessfully

processed entries are archived under /tmp/archive. Any failed records are

archived under /tmp/failed. Just like the mobile interface, processing events

in bulk will also cause the dashboard to automatically update.

Don't worry about making mistakes. The application is intended to be fairly

error tolerant. If you do come across issues, you should report them. Please

see the Getting Involved section on how to do so.

NOTE: All data entered is wiped upon application restart, so you can start from

a blank slate easily if needed.

You can also use the soapUI scripts included in the source code to explore the

REST interfaces as well as the numerous unit tests covering the code base

generally.

Exploring the Code

==================

As mentioned earlier, the real point of the application is demonstrating how to

create well architected, effective Java EE applications. To that end, once you

have gotten some familiarity with the application functionality the next thing

to do is to dig right into the code.

DDD is a key aspect of the architecture, so it's important to get at least a

working understanding of DDD. As the name implies, Domain-Driven Design is an

approach to software design and development that focuses on the core domain and

domain logic.

We have a brief overview of DDD specifically as it relates to Java EE on the

project site: https://java.net/projects/cargotracker/pages/Home. There's also a

resources page that you should take a look at:

https://java.net/projects/cargotracker/pages/Resources. The project site has

pages dedicated to explaining the core constructs of DDD and how they are

implemented in the application using Java EE:

https://java.net/projects/cargotracker/pages/Characterization as well as the DDD

layers in the application: https://java.net/projects/cargotracker/pages/Layers.

For the most part, it's fine if you are new to Java EE. As long as you have a

basic understanding of server-side applications, the resources referenced above

and the code should be good enough to get started. For learning Java EE further,

we have recommended a few links in the resources section of the project site. Of

course, the ideal user of the project is someone who has a basic working

understanding both Java EE and DDD. Though it's not our goal to become a kitchen

sink example for demonstrating the vast amount of APIs and features in Java EE,

we do use a very representative set. You'll find that you'll learn a fair amount

by simply digging into the code to see how things are implemented.

Known Issues

============

There are no known issues while running on GlassFish 4.1. For previous versions

you might run into the following issues:

\* If you restart the application a few times, you will run into a GlassFish 4

bug (https://java.net/jira/browse/GLASSFISH-20616) causing a spurious

deployment failure. While the problem can be annoying, it's harmless. Just re-

run the application (make sure to completely shut down GlassFish first).

\* You will see some spurious JSF warnings on some pages due to a GlassFish

4/Mojarra bug (https://java.net/jira/browse/GLASSFISH-20244). The error is

harmless and can be ignored.

\* Sometimes when GlassFish is not shutdown correctly, the Derby database that

the application uses get's corrupted, resulting is strange JDBC errors. If

this occurs, you will need to stop the application and clean the database. You

can do this by simply removing \temp\cargo-tracker-database from the file

system and restarting the application.

Getting Involved

================

Cargo Tracker is an open source project hosted on java.net. We would welcome any

and all contributions.

The project mailing lists are here: https://java.net/projects/cargotracker/lists

The JIRA issue tracker is here: http://java.net/jira/browse/CARGOTRACKER

You can also send an email to reza.rahman@oracle.com with any questions,

concerns or suggestions.