Deploying the application on any computer

The following code must be changed in our for the software to

1. Change the file path to your desired target directory at the following locations:

* *data\_import.jsp* - line 180
* *DirectoryWatcher.java* - line 26

1. Run the database creation file *lte\_failure\_system\_db.sql* to create the database structure with defined relationships.
2. Change the JNDI name in the *persistence.xml* to point to a database with database name *lte\_failure\_system*.

\**data\_import.jsp* is located in src > main > webapp > assets > pages >

\**DirectoryWatcher.java* Iis located in src > main > java > service > file >

\* *persistence.xml* is located in src > main > resources > META-INF >

Web service

We chose to develop our software using [WildFly](http://wildfly.org/about/) as implements the latest in enterprise Java standards.

Connection to the database

* The connection of our JEE project to our MySQL database is done through WildFly’s admin console using the JDBC (Java Database connector) and the defined JNDI (Java Naming and Directory Interface) ‘java:/lteFailureSystem’.

Entities

The most important and complex entity is *FailureEvent.java* class corresponds to the base data table in our database with ManyToOne relationships between itself and the lookup tables, as can be seen in the UML/Class diagrams.

To implement composite primary keys in Java JPA entities it was researched that using the following annotations allowed our software to map the relationships as required:

* @Embeddable: This is the composite primary key as defined in the corresponding lookup tables.
* @Embedded: This annotation allows us to embed the composite primary key into the JPA entity.

DAO Layer

The DAO layer has one job and that is to perform CRUD (Create, Read, Update, and Delete) transactions. They are relatively lightweight java classes and at the moment they are mainly used to check for valid relationships and for the one ‘Full Stack’ query we have implemented on the Front-End.

Business Layer

The Business layer contains all business logic that are associated with the requirements of the class itself. The Façade design pattern is used here to abstract the implementation and logic between the layers.

Important code elements currently located on this layer are as follows:

* *FailureEventBusinessImpl.java*
  + This class scans a csv file for each record in the base data.
  + For each row in the file the data is validated for correct datatypes.
  + If the data passes ‘datatype validation’ valid relationships are checked to ensure the record is a ‘valid Failure Event’.
* *DirectoryWatcher.java*
  + This class is responsible for ‘listening’ on a location, or in our instance polls a local directory for any changes that are associated with specific types of events. For this Sprint we have ‘*Entry\_Create*’ implemented. So, if a file is dropped into the directory our program will send the file
  + An important note about this class is that we used CDI to inject the EJB *(FailureEventBusinessImpl)*, this allows the program access the data validation element of this class allowing the program to automatically import valid data into the database.

REST Layer

The REST layer acts as an interface between the JEE layers and the Front-End. It handles HTTP requests such as POST and GET. At the moment the most important requests are as follows:

* ‘\*/file/dir\_watcher’: This returns all the data in the Failure Event table in our database. This query was important as it showed our software’s ability to perform a full stack query
* ‘\*/data/fail\_events’: This POST request starts our *DirectoryWatcher* and ensures that once a file is ‘dropped’ into a directory the *DirectoryWatcher* will recognise this event.