Architecture Specification Breakdown for Publications and Backend

Application server (Where we are working):

- Where business processes are deployed and executed. I.e. the various functions such as AddPublication() etc.

Refined quality requirements for application server:

- Flexibility:
  - Deploy versions of the system with minimum downtime.
  - Replace one lower level service with another e.g. persistence provider
- Reliability:
  - Service requests must be realised as per service contract (all pre-conditions must be met).
  - If a service is refused, a reason must be provided to the caller.
- Security:
  - Must support role-based authorization.
  - No direct access to the database (persistence provider), thus data is only accessible through services that you have created (get functions)
- Testability:
  - Out-of-container testing

Tactics:

- Flexibility:
  - Hot-deployment
  - Contract based development with dependency injection
- Maintainability:
  - Application server should be based on public standards
- Reliability:
  - Commit and rollback across persistence provider and mail server adapter
  - No hardcoding of transaction boundaries
- Security:
  - Specify which security role a user must have to use a particular service.
- Auditability:
  - Must enable logging
- Testability:
  - To reuse unit tests for integration and regression, frameworks must support dependency injection

Application Component Concepts and Constraints:

- Central concept used for application logic:
  - Service contracts: which describes the requirements for a service
  - Service which describes the implementation of the service
- Services will be stateless meaning that no states are kept between different requests.
- States are only maintained as domain objects within the persistence layer. (persistence api takes java objects and maps them to db)

Frameworks and Technologies:

- **Web Front End – Ember.js**
  - Integrates with JSON and RESTful applications
  - Do not use ORM directly make use of REST layer
  - *ORM = object relational mapping – used to create virtual object database

- **Java-EE 7 (Application Server NB)**
  - Flexibility/Maintainability:
    - decoupling through contracts/interfaces
    - CDI (Context and Dependency Injection) – java standard for dependency injection
    - Hot-deployment through JBoss and Glassfish
  - Scalability:
    - Concurrent users
    - Java-EE uses thread-pooling, object-pooling, connection pooling
  - Reliability:
    - Container-managed transactions across multiple transaction-supporting resources
    - Annotating methods with TransactionAttribute.Required specifies that service must run under transactional control
    - Read Architecture Specification for details, it gets complicated…
  - Security:
    - Java-EE supports role based authorization
    - Methods annotated with security roles required by users to use service
  - Auditability:
    - No explicit support
    - Allows intercepting all services with global interceptor which can be used to log service requests and results
    - EJB interceptor and Slf4jProvider (needs researching)
  - Testability:
    - Java-EE good support for dependency injection
    - Architecture not hard-coded within logic, but java-EE specific information is specified via annotations.
    - App code can thus be executed outside Java-EE container (for unit, integration, regression testing)
    - Dependency injection can be used to inject to inject mocks or actual dependencies (allow reuse of Unit tests)

Application component concepts and constraints:

Within Java-EE:

- Service contracts represented by Java interfaces
- Stateless services realized via methods of stateless session beans

    - Session beans do the work by executing the work by executing services from within the application server. Stateless beans' instance variables only have state during the duration of the execution. Basically a bean is a invocation of a method.
- Domain objects realized via Java entities (basically what will be stored in the DB)

Database:

PostgreSQL – Mature relational database

Persistence API (Java Persistence API):

Provides abstracted access to a database (persistence provider) while remaining decoupled

Persistence API should use:

- Object-relational mapping – maps java objects to database entries
- Query mapping – maps query of objects onto database queries
- Object caching
- Transactions

Application concepts:

- Domain objects: holds the long living states (only holds information)
- Queries across object graph of domain objects

Frameworks and technologies:

- JPA (Java Persistence API) will be used
- Default implementation package with chosen application server will be used (EclipseLink in the case of glass fish)
- Persistence context will be dependency injected into services requiring data
- JPA provider implements:
    - Object relational mapping via provided ORM
    - Query mapping
    - Object caching
    - Transaction support through **Java Transaction API**
    - Connection pooling through **JCA** based on implementation of a **JDBC driver**
    - Queries specified as JPQL (Java Persistence Query Language)

Overall view of the system:

View pdf named **SystemExplanaition.pdf,** image was created by Emilio Singh of Bravo Integration. It is meant to serve as a view of how everything fits together and what technologies fit in where.