

---

*Software Engineering for Web Applications*  
*Sample Example*

Denis Helic

- ▶ Lecture is not about
  - ▶ Web programming using a particular programming language
  - ▶ e.g. PHP, Java
- ▶ The Web today is an application platform
  - ▶ Therefore it is a subject to software application engineering

- ▶ Introduce an engineering method of developing Web applications
  - ▶ Collect requirements from users
  - ▶ Analyse requirements and decide on the server-side technology (PHP, Java, etc.)
  - ▶ Implement the system

## *Sample Application - Requirements(1/2)*

---

- ▶ Web-based database of scientific publications
  - ▶ Used for managing of personal publication databases
- ▶ Retrieving of publications in different formats
  - ▶ HTML, BibteX, XML
- ▶ Searching for specific publications
  - ▶ Type, year, title keywords

## *Sample Application - Requirements(2/2)*

---

- ▶ A special user as administrator
  - ▶ Can add authors
  - ▶ Can add publications
  - ▶ Delete, update!
- ▶ Importing publications in different formats
  - ▶ BibteX, XML

## *Selecting a server-side technology(1/2)*

---

- ▶ The Web is a specific application platform
  - ▶ HTTP is a connectionless protocol
- ▶ Crucial to track user sessions
  - ▶ Managing user information over multiple requests

## *Selecting a server-side technology(2/2)*

---

- ▶ PHP, Java provide high-level APIs for managing sessions
- ▶ CGI-scripts do not have such high-level interfaces
  - ▶ You as a developer need to take care of that
  - ▶ Create unique session IDs and make them persistent on the server side
- ▶ Go for PHP or Java ;)

## *Selecting a server-side storage(1/2)*

---

- ▶ Making application data persistent
- ▶ Relational Database Management Systems (RDBMS)
- ▶ File system
- ▶ Native XML databases

## *Selecting a server-side storage(2/2)*

---

- ▶ Go for RDBMS because
  - ▶ ACID test (Atomicity, Consistency, Isolation, Durability)
  - ▶ Declarative query language SQL
  - ▶ You tell the system what do you want not how to do it!
  - ▶ Mature products, know-how, experience, support, etc. (major problem of native XML databases)
  - ▶ More on native XML databases in MMIS2

## *Implementing the system - Engineering Method(1/3)*

---

- ▶ Pick a programming environment
- ▶ Develop a data model
  - ▶ What data do I need to store and how I will represent it?
  - ▶ Comes down to developing a database schema, i.e., database tables
- ▶ Develop a collection of legal transactions on the data model
  - ▶ How can I manipulate the data? (inserts, updates, selects, ... in SQL)

## *Implementing the system - Engineering Method(2/3)*

---

- ▶ Design the page flow
  - ▶ How do users interact with the system?
- ▶ Implement the individual pages
  - ▶ Accessing data
  - ▶ Wrap data in HTML

## *Implementing the system - Engineering Method(3/3)*

---

- ▶ The biggest issue of all Web applications
  - ▶ Mixture of data access (content) and presentation
- ▶ The goal to achieve
  - ▶ Separate content and presentation
  - ▶ People have done it before → Design Patterns!

## *Picking a programming environment(1/3)*

---

- ▶ Picking a RDBMS
  - ▶ Depends on requirements, know-how
  - ▶ Two choices: commercial and open source products
- ▶ For the sample example I selected MySQL
  - ▶ Open Source, free, nice documentation
  - ▶ Performance not that critical since personal database (requirements)

## *Picking a programming environment(2/3)*

---

- ▶ Picking an execution environment
  - ▶ CGI already gone because of session tracking
  - ▶ PHP or Java?
- ▶ Database application - intelligent managing of database connections (performance)
  - ▶ Both PHP and Java can have connection pools

## *Picking a programming environment(3/3)*

---

- ▶ PHP - scripting, interpreting, weakly type language
  - ▶ Fast to make the first implementation
  - ▶ Performance issues
  - ▶ Errors because of weak types, interpreter
- ▶ Java technologies
  - ▶ Personal preference
  - ▶ OO, compiled, I can use JSP additionally to have scripting!

- ▶ Managing of publications
- ▶ Each publication has
  - ▶ One or more authors
  - ▶ Title
  - ▶ Year of publishing
  - ▶ Optionally URL
  - ▶ Type (depends on format)

## *Developing a data model(2/4)*

---

- ▶ BibteX format <http://www.din1505.informationskompetenz.net/>
- ▶ Types such as Article, Conference, Book, etc
- ▶ Depending on type different additional attributes
  - ▶ Article has a journal
  - ▶ Book has a publisher, etc.

## *Developing a data model(3/4)*

---

- ▶ One table for publications
  - ▶ Keeps data common for all publications
- ▶ One table for persons involved
  - ▶ One publication can have many authors
  - ▶ One author can be involved in many publications
- ▶ One table which relates a publication with its authors
- ▶ One table which keeps additional attributes as key-value pairs

## *Developing a data model(4/4)*

---

- ▶ Always provide an SQL script which initializes the database
  - ▶ Make sure that a new user is created and privileges are assigned
- ▶ `http://localhost:8080/publicationdb/src/setupdb.sql`

▶ Analyse requirement to create queries

▶ Select all publications

```
select * from publication;
```

```
select person.peid, name, role from pubperson, person
```

```
    where pubperson.peid = person.peid and pubperson.pid = ?;
```

```
select * from pubattr where pid = ?;
```

 Select publications with a keyword in the title

```
select * from publication where title regexp ?;
```

```
select person.peid, name, role from pubperson, person
```

```
    where pubperson.peid = person.peid and pubperson.pid = ?;
```

```
select * from pubattr where pid = ?;
```

 Insert a new publication

```
insert into publication values(null, ?, ?, ?, ?);
```

```
insert into pubperson values(?, ?, ?);
```

```
insert into pubattr values(?, ?, ?);
```

- ▶ Which user classes do I have?
- ▶ “Normal” users
  - ▶ Can retrieve publications
  - ▶ Can use search form

- ▶ “Power” users
- ▶ Navigation frame
  - ▶ Can insert a new person
  - ▶ Can insert a new publication (need to pick authors)
  - ▶ Can update an existing person
  - ▶ Can delete an existing person
  - ▶ Can update an existing publication
  - ▶ Can delete an existing publication
- ▶ Authentication required

## *Implement the individual pages*

---

- ▶ SQL statement problems
- ▶ Authentication problems
- ▶ Connection pools
- ▶ Resolving content/presentation issue
- ▶ Supporting different output formats

## *SQL statement problems(1/4)*

---

- ▶ JDBC API provides the Statement class
- ▶ Statement objects usually applied in the following way

```
Statement statement = connection.createStatement();  
ResultSet result = statement.executeQuery(  
    "select * from publication where title regexp " +  
    request.getParameter("title"));
```

## *SQL statement problems(2/4)*

---

- Usually long statements → typing errors
- Security considerations
  - Suppose that users type “something ; select \* from passwords;”
- Performance issues
  - Each time the statement is compiled again!

## *SQL statement problems(3/4)*

---

 Solution: use PreparedStatement class

```
select_pubs = connection.prepareStatement(  
    "select * from publication where title regexp ?");  
select_pubs.setString(1, request.getParameter("title"));
```

## *SQL statement problems(4/4)*

---

- ▶ No typing errors
  - ▶ No need to compose the query string
  - ▶ Types are checked, e.g. `setString()`, `setInt()`
- ▶ The security issue resolved
  - ▶ Parameter is used as a value of the query variable
- ▶ Performance improved
  - ▶ `PreparedStatement` is pre-compiled only once

- ▶ DB authentication
  - ▶ Single DB user for all Web app users
- ▶ Application logic resolves Web app authentication
  - ▶ Which users can access what pages?

## *Authentication problems(2/7)*

---


- ▶ Types of Web app authentication
- ▶ Programmatic authentication
  - ▶ Authentication hard-coded in the scripts
- ▶ Declarative authentication
  - ▶ Declare username, passwords in a separate file
  - ▶ Let the servlet engine take care about authentication ;)

### Tomcat authentication

```
<security-constraint>
  <display-name>
    Publication Database Administration
  </display-name>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/administrator/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>publicationdb</role-name>
  </auth-constraint>
</security-constraint>
```

### Tomcat authentication (continued)

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Form-Based Authentication Area</realm-name>
  <form-login-config>
    <form-login-page>/jsp_utils/login.jsp</form-login-page>
    <form-error-page>
      /jsp_utils/login_error.jsp
    </form-error-page>
  </form-login-config>
</login-config>
<security-role>
  <role-name>publicationdb</role-name>
</security-role>
```

 Tomcat authentication (continued) tomcat-users.xml

```
<tomcat-users>
```

```
...
```

```
  <role rolename="publicationdb"/>
```

```
...
```

```
  <user username="pubadmin" password="admin"  
        roles="publicationdb"/>
```

```
...
```

```
</tomcat-users>
```

## *Authentication problems(6/7)*

---

➤ DB authentication usually implemented in the following way

```
Connection connection = DriverManager.getConnection(
    "jdbc:mysql://localhost/publicationdb",
    "username", "password");
```

➤ Problems

➤ Security, portability

## *Authentication problems(7/7)*

---

- ▶ A simple solution
  - ▶ Declare username and password as init-param in web.xml
- ▶ Another problem of the previous approach
  - ▶ Performance since each request opens a new connection
- ▶ A better solution
  - ▶ Resolve authentication issue together with connection pooling


- ▶ Broker class encapsulates access to the database connections
  - ▶ `getConnection()` method
- ▶ Behind the scene broker manages a buffer of connections
  - ▶ `getConnection()` returns the first available connection
  - ▶ If no free connection enlarge the buffer
  - ▶ When a client finishes it frees the connection (broker notified)
- ▶ Optimizing the buffer size!

- Usually JDBC drivers provide connection pooling
- Apache Tomcat provides connection pooling
  - Database Connection Pool (DBCP)
  - Part of Jakarta Commons project
  - `http://jakarta.apache.org/commons`

- ▶ DBCP uses Java Naming Directory Interface (JNDI)
- ▶ JNDI Data Source
- ▶ Define JNDI resource reference in web.xml
- ▶ Map JNDI resource onto a real resource in server.xml
- ▶ Lookup JNDI data source in the code

 Define JNDI resource reference in web.xml

```
<resource-ref>  
    <res-ref-name>jdbc/publicationdb</res-ref-name>  
    <res-type>javax.sql.DataSource</res-type>  
    <res-auth>Container</res-auth>  
</resource-ref>
```

 Map JNDI resource onto a real resource in server.xml

```
<Context path="/publicationdb" docBase="publicationdb"
  debug="0" reloadable="true" >
  <ResourceParams name="jdbc/publicationdb">
    <parameter>
      <name>username</name>
      <value>publicationdb</value>
    </parameter>
    <parameter>
      <name>password</name>
      <value>mmis2004</value>
    </parameter>
  </ResourceParams>
</Context>
```

.....

- ▶ Map JNDI resource onto a real resource (continued)

```
<parameter>
  <name>driverClassName</name>
  <value>org.gjt.mm.mysql.Driver</value>
</parameter>
<parameter>
  <name>url</name>
  <value>jdbc:mysql://localhost/publicationdb</value>
</parameter>
</ResourceParams>
</Context>
```

 Lookup JNDI data source in the code

```
Context init = new InitialContext();
Context ctx = (Context) init.lookup("java:comp/env");
DataSource ds = (DataSource) ctx.lookup("jdbc/publicationdb");
connection_ = ds.getConnection();
```

▶ Be careful!

▶ Need to notify the broker when finished

▶ Do so by closing connection and all other DB resources

```
try {  
    ....  
} finally {  
    close(insert_person, result);  
    closeConnection();  
}
```

## *Resolving mixture of content/presentation*

---

- ▶ Different desing approaches (patterns) to resolve the problem
- ▶ Model-View-Controller pattern
  - ▶ Applied in Struts framework (MMIS2)
- ▶ J2EE Design patterns
  - ▶ <http://java.sun.com/blueprints/corej2eepatterns/index.html>
- ▶ Data Access Object (DAO) pattern
  - ▶ <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>

## *Data Access Object - DAO Pattern(1/8)*

---

- ▶ Encapsualtes all database access into a single class
  - ▶ CRUD interface (create, read, update, delete)
- ▶ Works with transfer objects
  - ▶ TO reflect application logic

## *Data Access Object - DAO Pattern(2/8)*

---

▶ Example of DAO CRUD interface

▶ PersonDAO

.....

```
public void storePerson(Person person);
```

```
public Iterator readAllPersons();
```

```
public Person readPersonWithId(int id);
```

.....

```
http://localhost:8080/publicationdb/src/edu/iicm/publication/  
db/PersonDAO.java
```

## *Data Access Object - DAO Pattern(3/8)*

---

 TO Person encapsulates the app logic

.....

```
public Person(int id, String name);
```

```
public String getName();
```

```
public void setName(String name);
```

.....

```
http://localhost:8080/publicationdb/src/edu/iicm/publication/  
Person.java
```

## *Data Access Object - DAO Pattern(4/8)*

---

- ▶ Similar classes for Publication  
`http://localhost:8080/publicationdb/src/edu/iicm/publication/db/PublicationDAO.java`
- ▶ Publication class  
`http://localhost:8080/publicationdb/src/edu/iicm/publication/Publication.java`

## *Data Access Object - DAO Pattern(5/8)*

---

 Code to access the DAOs and present the results (Search interface)

```
.....  
String type = request.getParameter("type");  
String year = request.getParameter("year");  
String title = request.getParameter("title");  
Iterator pubs = dao.readAllPubs(type, year, title);  
while (pubs.hasNext()) {  
Publication pub = (Publication) pubs.next();  
out.println(pub.getStringRep(writer));  
}  
.....
```

## *Data Access Object - DAO Pattern(6/8)*

---

- ▶ Searching the database  
`http://localhost:8080/publicationdb/`
- ▶ Access code  
`http://localhost:8080/publicationdb/select_pubs.jsp.txt`
- ▶ Presentation code  
`http://localhost:8080/publicationdb/pubs.jsp.txt`

## *Data Access Object - DAO Pattern(7/8)*

---

- To decouple the storage from DAO class work with interfaces
- Work with Abstract Factory Pattern to obtain proper DAO instances
- Allows you to move to another storage, e.g., XML native database

## *Data Access Object - DAO Pattern(8/8)*

---

 Factory

`http://localhost:8080/publicationdb/src/edu/iicm/  
publication/db/DAOFactory.java`

 A particular implementation

`http://localhost:8080/publicationdb/src/edu/iicm/  
publication/db/PublicationJDBCDAOImpl.java`

## *Providing different output formats(1/3)*

---

- ▶ To have different output formats use Visitor pattern
- ▶ Publication is an abstract class with an abstract write method
- ▶ Subclasses (Article, Book, ...) invoke an abstract method from an abstract Visitor
- ▶ Subclasses of Visitor implement the method
  - ▶ Write out the proper format

## *Providing different output formats(2/3)*

---

- ▶ New format → new Visitor
- ▶ Creation of Visitor again Abstract Factory
- ▶ Allows you to use the same code to write out HTML, BibTeX, ...  
`http://localhost:8080/publicationdb/export.jsp`

## *Providing different output formats(3/3)*

---

- ▶ Interface  
`http://localhost:8080/publicationdb/src/edu/iicm/publication/writer/PublicationStringWriter.java`
- ▶ A particular implementation  
`http://localhost:8080/publicationdb/src/edu/iicm/publication/writer/PublicationStringHTMLWriterImpl.java`
- ▶ Factory  
`http://localhost:8080/publicationdb/src/edu/iicm/publication/writer/PublicationWriterFactory.java`

- ▶ Still under construction  
<http://localhost:8080/publicationdb>
- ▶ Administration interface  
<http://localhost:8080/publicationdb/administrator>
- ▶ Current build  
<http://coronet.iicm.edu/mmis/examples/sewa/publicationdb.war>