WEB - TEST 2

Web Services

- Information System: Presentation <-> Application <-> Resource Management
- Service: Business Functions an Enterprise offers to business Partners
  - ↳ Service Providers, Service aggregators, Service clients
  - ↳ Service Oriented Architecture: Pattern (abstract), loosely coupled, shared and reusable
- Web Services: self-contained, encapsulated functionality
  - contract
  - abstract underlying program logic
  - loosely coupled
  - reusable
  - may be composed
  - described with WSDL, WADL
  - distributed over the internet
  - ↳ Functional / Non Functional characteristics
  - ↳ Types: SOAP - WSDL / REST
  - ↳ Interoperable, Economical, Automatic, Accessible, Available
  - ↳ Publish, Find, Bind → WSDL, SOAP
  - ↳ SOAP, WSDL, UDDI → message exchange, description, directory service

- SOAP: envelope: what's in the message and how to process
  - encoding rules: for application data types
  - conventions: remote procedure call and response
  - -> using HTTP, SMTP, FTP... for transport
  - -> Envelope: Header: routing, delivery, authentification } application defined
    - Body: message payload
  - -> Nodes: transmit, receive, process, relay
  - -> Communication styles: remote procedure call (encoded / literal (XML Schema))
    - document exchange (encoded / literal)
      - ↳ may be asynchronous
  - -> Errors: use: Fault in body: code, Reason, Detail

- WSDL: what a service does, where it resides, how to invoke it
  - Service interface definition: supported operations, operation parameters, abstract data types
  - Service implementation part: concrete adress, concrete protocoll, concrete data structures
  - -> types / message, portType, operation, binding: soap:binding, operation
    - service: port, soap: adress

→ Document exchange: content of soap-Body defined by type element
→ RPC: soap-Body needs to comply with SOAP specification

Messaging pattern: port type → Operation → input / output
4 different patterns

- UDDI: Universal description discovery and integration,
  → defines schema for publishing and find web services
  ↳ white pages, yellow pages, green pages
  ↳ never really successful: discovery model limited
  pure technical terms
  Approach too generic
  Too many fun-entries

- Java Api: JAX-WS
  - Annotations,
  - contract first / contract last

- RESTful Web Services: Representational State Transfer
  ↳ centred around resources and resource states
  ↳ GET: receive, PUT: change, POST: create, DELETE: remove
  → URI's: address resources

- Java API: JAX-RS
  - Annotations

→ HEAD: only GET header
→ OPTION: describes the web service: WADL

Linked Open Data

- Open Data: Availability and Access
  Reuse and Redistribution
  Universal Participation

  ↳ Open Government Data: Complete — Non discriminatory
  Primary — Non. proprietary
  Timely — Licence free
  Accessible — Permanent
  Machine Readable — Freely obtainable

- Linked Data: Everything is identified via URIs
  Content refers to semantic metadata
  Ontologys used for knowledge representation

  ↳ Vision: machine readable (semantic web)
  ↳ Semantic web: Use HTTP uris for everything
  Provide useful information on URL
  Information should be discoverable

  ↳ Ontologies: explicit formal specification of a shared conceptualization
  ↳ common understanding of what information means

  → RDF: resource description framework
  - subject, predicate, object

  → SPARQL: query data

JPA and Hibernate

- JDBC: access relational databases from Java
  - establish connection
  - execute statements
  - create parameterized queries
  - manage database connections

- Object Relational Mapping: - automatic synchronization between Java Objects and underlying database
  - DB independent
  - Query abstraction

- JPA: Specification for object / relational mapping in Java

  → map Java Objects to database → Objects outlive the JVM
  → Hibernate: Full JPA implementation with additional features:
  ↳ HQL, Criteria API

↳ Simple Mapping: @Entity, @Id, @Temporal, @TemporalType, @Transient
@MappedSuperclass, @GeneratedValue

↳ Entity relationships: specify owning side (→ store foreign key on owning side)

- OneToOne: Embedded Table: @Embedded, @Embeddable
  Bidirectional: @OneToOne; mapped by, @JoinColumn: owning
- OneToMany: Bidirectional: @OneToMany, @ManyToOne: owning
- ManyToMany: @ManyToMany: mapped by

→ Cascade: Cascading operations to associated entities (cascade=)
ALL, PERSIST, MERGE, REMOVE, REFRESH, DETACH

→ Fetching: how object hierarchies loaded (fetch=)
EAGER, LAZY

↳ Persistence Concepts: Persistence Unit: set of classes, managed by entity manager
Persistence Context: set of entity instances, runtime
Entity manager: API for interaction with Persistence Context

→ Entity manager: persist(), remove(), refresh(), merge(), find(), contains(), flush(), createQuery(), createNativeQuery()

→ JPQL / HQL: abstract from vendor specific SQL
→ Dynamic Query
→ Static Query: Named

WEB 2.0

↳ Internet based communication and collaboration tools

↳ Content Typically user generated

↳ Many new existing technology stack is used

Angular JS

↳ Move MVC architecture to the client side
  ↳ Enable single site web applications
  ↳ Separation of logic: controller, data models, views

- Directive: defines new HTML-attribute that has custom functionality
  ↳ matching: various possibilities to match directive
    ↳ normalization

  -> Some directives override HTML-Elements, other are clearly visible

- Data Binding: automatic synchronization between view and model

- Scopes: context, custom behaviour and data
  determine execution context for expressions

- Expressions: JS-like code snippets
  evaluated against scope, no control functions

- Controllers: set up and add custom behaviour to scope
  -> ng-controller directive
  -> nested controllers

- Modules: containers for controllers, services, filters, directives
  -> keep global namespace clean

- Services: create independent application logic
  -> singletons, lazy instantiation per web-application

- Filters: formats the values of an expression
  -> stateless, idempotent

↳ $http service: $promise, promise (pending / fulfilled / rejected)

↳ $apply function: re-run the $digest-loop

Web Engineering @ Work

- no cloud services, custom CMS „Mega"
  ↳ build upon Helma and Ringo JS

- 90 external servers, same type of machines
  ↳ debian

- Oracle and PostgreSQL

- HTML: avoid errors in validator
  HTML5: live streams complicated

- CSS: SASS, SCSS
  box sizing: border-box
  relative units, avoid floats

- Fonts: custom font family

- Accessibility: Perceivable, Operable, Understandable, Robust
  ↳ avoid tricks if possible

- Responsive: don't do mobile first
  Problem with ads

- Javascript: don't over-engineer

- General: JSON over XML
  Apps: go native if important

- Project management: JIRA ticket management
  Slack
  Email

- Wahlcomputer: client side web application

- Globus: rendering of geo-information

AKTIONSGEMEINSCHAFT