Web Services Development Study Notes

Web Concepts

- **HTTP Methods:** Is a request/response protocol between client and server.
    - **GET:** Is a request representation of a specific resource. It is idempotent and can take parameters through the URL.
    - **POST:** Submits data to be processed. It is non-idempotent and can take data through the body of the request.
    - **PUT:** Submits data to be processed. It is idempotent, unlike POST, so it is ideal for update actions.
    - **DELETE:** Removes a resource.

- **Web Apps:** Presentation-oriented, page-based sites for human interaction that uses markup to display information. Interactions are request/response.
- **Web Services:** Service-based, machine-oriented sites for interaction with other applications with remote methods. Interactions are request/response or messages.
    - **Representational State Transfer (REST):** Is a design pattern for creating web services, based on the concept of resources that map well to database CRUD (Create, Read, Update, Delete) actions. This leads to the design of resource-oriented architecture (ROA).
        - Principles
            - Every resource must have an ID, otherwise known as a Uniform Resource Identifier (URI)
            - Resources are linked where possible
            - They use Standard HTTP methods such as GET, POST, PUT, DELETE
            - Potentially offer multiple representations (e.g. JSON, XML, CSV)
            - The resources remain stateless
        - **URI Guidelines:** Suggest that language should describe the resource, use a like path for like resources, and minimise the use of query strings (aka. “/rest/user?name=bob”).
    - **Simple Object Access Protocol (SOAP):** Is a protocol for transferring data from one application to another that encapsulates Remote Procedure Calls (RPCs) into messages, using existing, recognised standards in data representation and transport. While REST services are easier to conceptualise and easier to develop, SOAP has greater standards for security and transactions, and can work over non-HTTP protocols, making it a robust solution for distributed computing platforms.

- Message Components
    - **Envelope:** Describes the message and how it should be processed.
    - **Header:** Contains the features of the message including intended recipient.
    - **Body:** Contains the primary information to be received.
- **Web Services Description Language (WSDL):** Is an XML-based description of available web services.
    - WSDL Components
        - **Service:** Refers to a collection of endpoints, each being a combination of a port and binding that groups related ports together.
        - **Port:** Specifies the location of the web-service.
        - **Binding:** Maps operations to a protocol, like “SOAP via HTTP”.
        - **Port Type:** A set of functions in an interface supported by the endpoints.
        - **Operations:** Abstract descriptions of a supported action.
        - **Message:** A definition of a set of parameters.
        - **Types:** A definition of the usable data types.
- **Universal Description, Discovery and Integration (UDDI):** Serves as a directory for consumer services to index service provider details. The provider publishes its details to the UDDI, from where the client can find them.

- **Extensible Markup Language (XML):** A markup language for storing and transporting data, designed to be both human and machine-readable as it is self-descriptive. Unlike HTML, it was designed to carry data, rather than simply display it, and its tags are not predefined. The are plain-text documents being document-centric, and perform data storage and exchange duties being data-centric.
    - **Well Formed Documents:** Comply with the basic rules of XML. Being that they:
        - Have a root element
        - Have a closing-tag to every start-tag
        - Have all tags in-order
        - Are syntactically correct
    - **Valid Documents:** Are well formed and comply with their DTD constraints; schema valid.

- **Tags:** Define the data contained in the document or how they should be interpreted.
  - **Comments:** are ignored by processing:

    -
      ```
      <!--  some text description -->
      ```
  - **Prolog:** Must come first as they define the XML version and character encoding:

    -
      ```
      <?xml version="1.0" encoding="UTF-8" ?>
      ```
  - **DTD Declarations:** Document Type Definitions specify semantic constraints:

    -
      ```
      <!DOCTYPE exampleApp SYSTEM "./def/exampleApp.dtd">
      ```
  - **Elements:** Contain data values or other XML elements. They can also have attributes, formatted as name-value pairs:

    -
      ```
      <user type="admin">
          <name>John</name>
      </user>
      ```
  - **Entity References:** Are value identifiers, often used to refer to reserved characters like "<". They can also be defined by the user:

    -
      ```
      &amp; <!-- resolves to an ampersand (&) -->
      ```

    -
      ```
      <!ENTITY tst "this is a test value">
      <example>&tst;</example>
      ```
  - **Processing Instructions:** Tell applications what to do:

    -
      ```
      <? display table-view ?>
      <? sort alpha-ascending ?>
      ```
  - **Unparsed Character Data:** Tells XML processor not to parse the information:

    -
      ```
      <![CDATA[out.print(4 + 5)]]>
      ```
- **Namespaces:** Can be used to avoid naming clashes between documents and elements, and must be declared at the root element:

  -
    ```
    <Example:exampleApp xmlns="http://www.xyz.com/example">
    </Example:exampleApp>
    ```

- **XML Document Structure:** This allows for XML documents to be validated against a defined structure of rules.
  - **Document Type Definition (DTD):** The older definition language as it does not support custom data types, or any data-types aside from text for that matter, e.g.

    -
      ```
      <!ELEMENT example (id)>
      <!ELEMENT id (#PCDATA) #REQUIRED>
      ```
  - **XML Schema Definition (XSD):** Is the more modern standard and allows for custom data types on top of a number of base formats. It is a sub-language of XML often represented by a separate document with the extension .xsd. It defines the elements and attributes that may appear, the number and order of child elements, the data types for those elements and attributes, and the default or fixed values for them. This assists with guaranteeing the: correctness, restrictions on, and formats of data, making it easier convert data types.
    - **Types:** string, (integer, positiveInteger, negativeInteger), (base64Binary, hexBinary), decimal, anyURI, boolean, (time, date, dateTime, duration), Name, QName, (ID, IDREF)

```
<!— Elements —>
<xsd:element name=“id” type=“xsd:integer” minOccurs=“1” maxOccurs=“1”>

<!— Attributes —>
<xsd:attribute name=“type” type=“xsd:string” [options]>

<!— Complex Elements —>
<xsd:complexType name=“user”></xsd:complexType>

<!— Example Document —>
<!— Specifies the XML version —>
<?xml version=“1.0” ?>
<!— All docs begin with this element to define elements and namespace —>
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”
            targetNamespace=“http://example.com/example”
            xmlns=“http://example.com/example”
            elementFormDefault=“qualified”>

    <!— Define a custom data type with a regex pattern —>
    <xsd:simpleType name=“pin”>
        <xsd:restriction base:xsd=“integer”>
            <xsd:pattern value=“[09]{4}”>
        </xsd:restriction>
    </xsd:simpleType>

    <!— Define a user as having a number of sub-elements —>
    <xsd:element name=“user”>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name=“id” type=“xsd:integer”/>
                <xsd:element name=“username” type=“xsd:string”/>
                <xsd:element name=“pin” type=“pin”/>
            <xsd:sequence>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```