Hibernate :

1. **What is hibernate filter ?**

Hibernate 3 provides an innovative approach to handling the data with visibility rules. Hibernate filter is a global , named ,parameterized filter that can be enabled or disabled for the hibernate session. It allows hiberate to predefine filter criteria and apply that filter to class level or collection level. It is used to form a restriction clause similar to where clause. It is an alternative approach to DB views , where clause and Criteria API. It is similar to DB Views executed in application level and its parameterized. Filter can be defined by using the XML or Annotations. The application can decide at runtime which filter needs to be enabled and what is the value to be applied.

Filter definition has name and array of parameter. The parameter has name and type.

1. **Using the Filter in XML :**

The filter needs to be defined in hibernate mapping tag and can be applied to class or collection.

Ex:

The following filter , filter the data based on the given date.

<hibernte-mapping>

**// filter definition**

<filter-def name=”stockRecordFilter”>

<filter-parameter name=”stockRecordFilterParam” type=”date” />

</filter-def>

<class name=”com.stock” table=”stock” >

<set name=”stockDailyRecords” table=”stock\_daily\_record”>

<key column=”STOCK\_ID” not-null=”true” />

<one-to-many class=”com.StockDailyRecord” />

**//apply define filter to the colletion**

<filter name=” stockRecordFilter” condition=”stDate>=: stockRecordFilterParam” />

</set>

</class>

<hibernte-mapping>

Condition =”stDate>=: stockRecordFilterParam” , stDate is the proeprties from StockDailyRecord , stockRecordFilterParam is an filter parameter value to be set at runtime by an application.

1. **Using the Filter inAnnotations :**

The filter definition should be define by using the @FilterDef from org.hibernate.annotations.FilterDef or org.hibernate.annotations.FilterDefs. The filter param should be defined as @ParamDef.

Ex:

@Entity

@FilterDef(name=”stockRecordFilter” parameters=@ParamDef(name=”stockDailyFilterParam” type=”date”)

@Table(name=”stock”)

Public class Stock implements java.io.Serializable{

@one-to-many(fetch=FetchType.LAZY,mappedBy=”stock”)

@Filter(name=” stockRecordFilter” condition=”stDate>=:stockRecordFilterParam”)

Set<StockDailyRecord> stockDailyRecords;

Public Set<StockDailyRecord> getStockDailyRecords {

Return this.stockDailyRecords;

}

}

Code to use filter using the hibernate session,

Filter can be enabled and disabled by using the enableFilter() and disableFilter() methods available in hibernate session.

Session session=sessionFactory.getSession();

// Enable the session

Session.enableFilter(“stockRecordFilter”); // which returns the filter Instance

Stock stock=session.get(Stock.class,2);

//this set has the filter applied

Set <StockDailyRecords> sets=stock.getStockDailyRecords();

Session.diableFilter(“stockRecordfilter”)

//clean up the session

//retrieve the stock records now

Stock stock=session.get(Stock.class,2);

//this set doent have the filter applied

Set <StockDailyRecords> sets=stock.getStockDailyRecords();

In the above example, the same filter can be applied to class level also.

Another Ex:

@FilterDef(name="studentFilter", parameters={

@ParamDef( name="maxAge", type="integer" ),

@ParamDef( name="minAge", type="integer" ),

@ParamDef( name="minNumber", type="integer")

})

@Filters( {

@Filter(name="studentFilter", condition=":minAge <= age and :maxAge >= age"),

@Filter(name="studentFilter", condition=":minNumber <= number")

} )

In this above ex, there are 2 conditions are applied to same filter named. When use more than one @filters use the @Filters as like above.

**Using the filters in the jointable:**

1. **What is query caching in hibernate?**

Query result can also be cached in hibernate. This is useful for the queries that are frequently validated with same parameters. Query caching should be used with secone level cache.

Query cahcing is not enabled by default.

To enable the query caching we have to do the following change in config file,

Hibernate.cache.use\_query\_cache=true;

This setting creates two cache regions,

1. Org.hibernate.cache.internal.StandardQueryCache – cache the query result
2. Org.hibernate.cache.spi.UpdateTimeStampCache – hold the timestamp for the most recent updates to queryable tables.

To use the query caching , set org.hibernate.Query.setCahable(true), This will look for the result in cache , if not find execute the query on database and add the results to the cache. The query cache regions can be named by a specific name.It can be achieved by setting the org.hibernate.Query.setRegion(“regionname”. This allows the query caching to referesh only the particular region of the cache. To refresh the Query cache region use org.hibernate.Query.setCacheMode(CachMode.REFRESH).

**Ex:**

List list=session.createQuery(“from employee where empno>=?)

.setInt(0,45);

.setCachable(true)

.setRegion(“emplist”)

.setCacheMode(CacheMode.REFRESH)

.list();

**How its represented inside the query cache:**

The combination of query and the parameters passed to that query used as a key , and the identifiers of the result used as value.

[“from employee where empno>=?,[45]] - > [2,4]]

2 and 4 – are an identifier of the employee entity , by using this id it will load the actual entity from the cache.

1. **What is restful web services ?**

**REST** stands for Representation State Transfer . Restful webservices are Rest architecture based web services . In Rest architecture every content is treated as resource . These resources can be accessed via URI. Unlike SOAP webservice , it doent have its own security standards instead it derive its from underlying protocol. It use the HTTP protocol for data communication. Refer the web application security standards to make the restful services secure such as , Input validation, session based authentication , no sensitive data in the uri like user name , password and session tocken , Restriction on method execution, validate malformed XML and JSON , throw generic error messages.

Restwebservices are lightweight (no extra xml markup required) and highly scalable and easy maintanable and easy to build(no toolkits required) . These are used to create API’s for web based application. There are various representations of resource supported as XML,plain text and JSON. The most popular representation used are XML and JSON. The resource is REST similar to Object in Object oriented programming or Entitty in the database.

Rest Server simply give access to the resources , rest client can access and modify the resources.

The following Http methods commonly used in the Rest web Services.

**Get –** It is provide the read only access to the resource

**Put-**  it is used to create the new resource

**POST-** it is used to update an exisiting resource or create the new resource

**Delete –** it is used to remove the resource

**Options –** it is used to get the supported operations on the resource

**Ex:**

package com.tutorialspoint;

import java.util.List;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/UserService")

public class UserService {

UserDao userDao = new UserDao();

@GET

@Path("/users")

@Produces(MediaType.APPLICATION\_XML)

public List<User> getUsers(){

return userDao.getAllUsers();

}

}

@path – is used to specify the path of the web service and resource

@Path(“/UserService”) – is the path of the web service to be accessed

@Path(“/users”) – is the path of the resource to be accessed

@GET – is the method to be used to access the resource

@Produces - Specifies the MIME media type(data format) of the response that the resource can produce and send back to the client .

@Produces(MediaType.APPLICATION\_XML,MediaType.APPLICATION\_JSON) – it produces XML response also JSON response. @Produces applied to both class level and method level. If it is applied to class level all the methods return the same MIME media type specified at class level. If it is applied at method level , it overrides the class level settings.

The above web service can be accessed via

http://localhost:8080/UserManagement/rest/UserService/user

protocol://hostname:portnumber/servicename /urlpattern/resourcetype / resourceid

**Idempotent :-** Results will be same always howmany times we hit the same request.

In the above Ex, getUsers() method is accessing the userDao.getAllUsers()

Lets define the UserDao here,

package com.tutorialspoint;

import java.util.ArrayList;

import java.util.List;

public class UserDao {

public List<User> getAllUsers(){

List<User> userList = null;

//DB login to fetch the uswer from database , for temporarily just construct the User here

User user = new User(1, "Mahesh", "Teacher");

userList = new ArrayList<User>();

userList.add(user);

return userList;

}

}

From the above code , the getAllUsers() method constructing the list of User and returning it.

Lets construct the User object or Entity,

package com.tutorialspoint;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "user")

public class User implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String name;

private String profession;

public User(){}

public User(int id, String name, String profession){

this.id = id;

this.name = name;

this.profession = profession;

}

public int getId() {

return id;

}

@XmlElement

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

@XmlElement

public void setName(String name) {

this.name = name;

}

public String getProfession() {

return profession;

}

@XmlElement

public void setProfession(String profession) {

this.profession = profession;

}

}

The above entity itself annotated with @XmlRootElement and @XmlElement. These are JAXB annotations as described below

@XmlRootElement(name=”name” or namespace=”namespace”) - specifies the Root element of the XML. It is the class level annotation

@XmlType(proporder={field1,field2}) – specifies the order of the element in the XML. It is the class level annotation

@XmlElement (name=”eleemntname” )– specifies the element of the XML. it is the method level annotation.

Ex:

package de.vogella.xml.jaxb.model;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

**@XmlRootElement(name = "book")**

// If you want you can define the order in which the fields are written

// Optional

**@XmlType(propOrder = { "author", "name", "publisher", "isbn" })**

public class Book {

private String name;

private String author;

private String publisher;

private String isbn;

// If you like the variable name, e.g. "name", you can easily change this

// name for your XML-Output:

**@XmlElement(name = "title")**

public String getName() {

return name;

}

// other getters and setters

}

As per REST architecture, restful web services are stateless.

**JAX-RS** is a java based API useful for creating the Rest ful web services and clients. This API supports various annotations as below for restful web service creation.

1. **@Produces -** specifies the response type of the resource (web service method)
2. **@Consumes –**specifies the request type to be consumed by the resource
3. **@QueryParam –** it is used to inject the url query string parameter with the method parameter

Ex: “**users/query?from=100&to=200&orderBy=age&orderBy=name**”

import java.util.List;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.Response;

@Path("/users")

public class UserService {

@GET

@Path("/query")

public Response getUsers(

**@QueryParam("from")** int from,

**@QueryParam("to")** int to,

**@QueryParam("orderBy")** List<String> orderBy) {

return Response

.status(200)

.entity("getUsers is called, from : " + from + ", to : " + to

+ ", orderBy" + orderBy.toString()).build();

}

}

1. **@PathParam –**  it is used to inject the parameter value from the uri, that is defined in the @path expression into the java method arguments.

Ex:

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.core.Response;

@Path("/users")

public class UserRestService {

@GET

@Path("**{year}/{month}/{day}")**

public Response getUserHistory(

**@PathParam("year")** int year,

**@PathParam("month")** int month,

**@PathParam("day")** int day) {

String date = year + "/" + month + "/" + day;

return Response.status(200)

.entity("getUserHistory is called, year/month/day : " + date)

.build();

}

}

1. **@FormParam –** It is used to inject the HTML form parameters to the java method arguments.

**Ex:**

import javax.ws.rs.FormParam;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.core.Response;

@Path("/user")

public class UserService {

@POST

@Path("/add")

public Response addUser(

**@FormParam("name")** String name,

**@FormParam("age")** int age) {

return Response.status(200)

.entity("addUser is called, name : " + name + ", age : " + age)

.build();

}

}

1. **@CookieParam :-** it is used inject the entire cookie or cookie parameter value with the method.

**Ex: 1**

@GET

**public** Response getCookie(@CookieParam(**"name"**) Cookie cookie){

**return** Response.ok().build();

}

@GET

@Path(**"/value"**)

**public** Response getCookieValue(@CookieParam(**"name"**) String cookie){

**return** Response.ok().build();

}

1. **@HeaderParam :-** It is used to inject the specific header value to the method parameter. It can also been done by @Context ,but @Context HttpHeaders inject the entrie header into method.(refer @context down)

**Ex:**

@Path("/users")

public class UserService {

@GET

@Path("/get")

public Response addUser**(@HeaderParam("user-agent")** String userAgent) {

return Response.status(200)

.entity("addUser is called, userAgent : " + userAgent)

.build();

}

}

It prints the user-agent header value.

1. **@MatrixParam:-** It is used to inject the parameter value comes along with path but separated by semicolon. It is an key-value pairs

**EX:** “**/books/2011;country=malaysia;author=mkyong**”

@Path("/books")

public class BookService {

@GET

@Path("{year}")

public Response getBooks(**@PathParam("year")** String year,

@**MatrixParam("author")** String author,

@**MatrixParam("country")** String country) {

return Response

.status(200)

.entity("getBooks is called, year : " + year

+ ", author : " + author + ", country : " + country)

.build();

}

}

Year =2011

Author=mkyoung

Country=malaysia

1. **@Context :-** it is an alternative method of @QueryParam used to inject the paramter value from URI to method local variable. It is also called as programmetic way of receiving the uri query parameter. Basically it retrieves the context of the resource ie. Request context. The entire URI is injected to method. It is also used to inject the http header into method.

**Ex: 1**

@Path("/users")

public class UserService {

@GET

@Path("/query")

public Response getUsers**(@Context UriInfo** info) {

String from = info.getQueryParameters().getFirst("from");

String to = info.getQueryParameters().getFirst("to");

List<String> orderBy = info.getQueryParameters().get("orderBy");

return Response

.status(200)

.entity("getUsers is called, from : " + from + ", to : " + to

+ ", orderBy" + orderBy.toString()).build();

}

}

**Ex: 2**

@GET

@Path("/get")

public Response addUser(@**Context HttpHeaders** headers) {

String userAgent = headers.getRequestHeader("user-agent").get(0);

return Response.status(200)

.entity("addUser is called, userAgent : " + userAgent)

.build();

}

1. **@DefaultValue :-** It is used to specify an default value for the optional parameter.

**Ex:**

@Path("/users")

public class UserService {

@GET

@Path("/query")

public Response getUsers(

**@DefaultValue("1000")** @QueryParam("from") int from,

**@DefaultValue("999")@**QueryParam("to") int to,

**@DefaultValue("name")** @QueryParam("orderBy") List<String> orderBy) {

return Response

.status(200)

.entity("getUsers is called, from : " + from + ", to : " + to

+ ", orderBy" + orderBy.toString()).build();

}

}

1. **@Path :-** specifies the path to the resource and web service
2. **@Get :-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users> - retrieves list of users

<http://localhost:8080/UserManagement/rest/UserService/users/1> - retrieve the user of id 1

1. **@ Post :-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users/2> - Update the user with id 2

1. **@Delete:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users/1> - delete the user with id 1

1. **Put:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users/2> - Insert he user with id 2

1. **Options:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users> - list out the supported operations in web service

1. **@Head:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users> - retrieves http header

**Advantages :**

1. **Caching :**

Rest provides good caching mechanism.

**What is caching ?**

Caching means storing the server response in the client side. So that client no need to make the subsequent request for the same resource again and again. The server response itself should have information how to cache the response. So that this would be cache for the time period or not cached .

The following the headers available in HTTP server response with respect to caching.

**Date :-** The date when the resource is created.

**Last Modified:-** The date and time when the resource is modified at last

**Cache-Control:-** Primary header to control caching

**Expires:-** Expiration date and time of caching

**Age:-** Duration in second from when the resource is fetched from server.

**Cache-Control Headers:**

**No-cache/ no store : -** indicates the resource is not cashable

**Private :- I**ndicates that only that particular client and server can cache the resource not the intermediary

**Public :-** indicates that any one can catch the resource

**Max-age:-** indicates that the validity of the cached resource in seconds. After this the client can make another request

**Must-revalidate:**- indicates to server to revalidate the resource if max-age has passed.

**Ex:**

Tutorials point web site caching the pages and when we hit the send time (without network connectivity) still it shows the cached pages.

1. **Stateless:**

Restful web services provide stateless invocation. So the server not required to store anything across any request. So There will not much communication between server makes it horizantally scalable (refer horizantal scaling below). So that load balancer easily route the request to different servers.(Get request into one server and Post request into other server without replicating the session states among these servers). But SOAP relies on state and session , so we have to use the smarter load balancer to redirect the client request for process or should replicate the sessions state among the server . So SOAP is for stateful invocation.

1. **High Scalabiltiy:**

Because of the caching and statelessness behaviour of the Restful web services make this highly scalable in horizantally.

1. **Lightweight:**

Restful webservices support the plain text , xml and JSON message formats. It doent wrap the messages with some other standard . But SOAP web services , the xml messages are wrapped into SOAP Envelope (SOAP standard) which adds another layer on the message. It makes SOAP as heavyweight.

1. **Support of multiple data formats:**

Restful web services supports multiple data formats such as XML, plain text and JSON .

1. **Fast Development:**

It is easy to develop , no toolkit required for development and learning curve is very short. But SOAP is difficult to implement , it requires toolkit for development. Good learning is required about SOAP standards.

1. **Good Performance:**

Since its light weight and the result are human readable form , its giving good performance.

But in SOAP the result are not human readable form , it needs serialization of converting the xml to object and object to xml.

**Disadvantages:**

Restful web services has some disadvantages over SOAP.

1. **Security:**

It doesn’t have its own standards for security. It always rely on http standars on security. But SOAP has Security standards as WS-Security

1. **Protocol Support:**

It supports only HTTP protocol. Tied to HTTP transport protocol .But SOAP is designed to support for multiple protocols such as HTTP ,TCP , UDP and SMTP.

1. **Error Handling:**

It doesn’t have its own error handling standards. It depends on HTTP error handling standars . But SOAP have its own error handling standard as SOAP Fault.

1. **When to use REST web service over SOAP ?**

Choose SOAP :

1)When we have formal contracts

2) When stateful conversations are required

3) Asynhronous Processing and invocation

4) Learning curve takes time

5) it needs toolkit for development efficiency(WSDL dev)

6) When application needs end-to-end security or absolute distributed transational reliability

7) when only XML can be supported

Choose Rest:

1. When Limited bandwidth and resources
2. When need statelesss processing
3. Caching situations
4. Learning doesn’t take much time
5. It doesn’t require the tool kit for development
6. It supports only point-to-point communication.
7. Multiple data formats to be supported
8. Rest supports high scalability than SOAP(Restricted get is supported but not in SOAP) for web application with high load.
9. Rest messages are smallar in size and consume lesser bandwidth
10. Rest provides better caching so it increase performance
11. Errorhandling

Both are reliable and flexible approach to expose the functionality in distributed environment. Both has advantages and disadvantages .But still we need to conside the below two points whne selecting the best approach.

1. Type of clients supported. Rest provides an effective way for interacting with lightweight clients as mobile phones
2. Flexibility and standardization acccepted by the organizations culture.
3. Architecture requirement (end-to-end or point-to-point)
4. **What are micro services?**
5. **How to scale our application ? What is horizontal scaling and vertical scaling?**

**Scalability: Capacity of the system**

**Horizontal scaling:** Horizontal scaling means adding more machines into the pool of resources. The productivity of each machine or node or actor is still same but the capacity is increased by adding more machines or actors or nodes. It depends on the network capability.

**Vertical scaling:** Vertical scaling means adding more power into an existing machine. Adding resources to a single node in the system. The capacity of the system is increased by adding more power(CPU) to an existing machine or actor or node.

1. **What is reengineering? What are the aspects needs to be considered while reengineering the application?**

Reengineering means restructuring or rewriting the part of the system or the entire system without affecting the existing functionalities. The reengineered system must be redocumented.

**When the system needs to be reengineered:**

* When hardware or software’s are out dated
* When the system requires frequent maintenance
* When one component or system changes affects the other system or component
* When the tools required for reengineering is readily available

**Why do we need to reengineer the system :**

* When we need to reuse the existing logic(Code base) and functionality of the legacy system , we can go for reengineering of the system. So the cost and effort spend for building the old system is not wasted.

**Objective of the reengineering:**

* Improve the maintenance
* Access to the new platform
* Enhancing the functionality
* Improved reliability

**How to reengineer:**

* Analysis the existing system
* Find out the characteristic and feature of the target system
* Create the standard set of tests to conform the transformation
* Reengineering starts with the code base of the existing systems and ends with target system source code.
* Reengineering is very complex since we have to analysis the existing code , determine the requirements , compare them with current requirement s, remove unwanted things , design the new system and code.

**Reengineering approaches:**

* There are three approaches available in software reengineering such as

1. **BIG-BANG approach:**

It replaces the entire system into new system(moving the system to different architectures)

The advantage is entire system is moved to new environment and new interfaces needs to be developed. There will not be any old piece.

Disadvantage is results are not appropriate. For large systems it requires more resources or requires large amount of time to generate the new system

1. **Added Method approach:**

It is also called phase out. It this approach , only part of the old system needs to be reengineered and moved to new system. And added all new incremental updates into the new system.

The advantage is the components of the system are produced faster. It takes less time to create the new component when compared to the first approach. And it is easy to control. Since its an interim version , customer can see the progress and easily identify the gaps. This approach has lower risk when compared to first approach. The risk of the code to be identified and tracked.

Disadvantage is whole system will not be changed only the components has changed. This requires careful identification of the components of the existing system

1. **Evolutionary approach:**

This is called as method of evaluation. The part of the system is reengineered into new system. The part of the system is selected based on the functionality not based on the structure or component. The new system is built based on the functionality. The components of the current system is divided by function and the new system is created.

The advantage is, this approach gives modular design and well suited for the object oriented technology.

Disadvantage is same function must be identified in different components of the current system.

**Stages involved in the Reengineering:**

There are five stages involved in the reengineering.

1. **Establish Reengineering:**

In this stage, the project manager is responsible to handle the technical challenge and maintenance. They are also responsible for identifying, purchasing and testing the new tools or software that are required for the reengineering. The have to make sure that all employees (team members) are get trained well in the new technology and use them effectively. Though the team members have the software knowledge , they have to get trained in the new technology.

1. **Analyze the feasibility of the project:**

The reengineering team must assess the needs of the organization and the goal of the new system. They have to properly analyze the value of the current system and determine what has to be improved in the new systems such as maintenance efficiency , maintenance cost , software quality and value of the current system.

1. **Analysis and Planning:**

This involves 3 steps:

1. Current system characteristic, specification and quality needs to be identified. This starts with identifying the requirement documents, functional documents, design documents , use cases . These are needs to be helpful to determine the direction. Once the this is done , current status of the current system , maintenance needs to be determined for the capital of the target system. Then set of software metrics needs to be set and asses the current system to determine the quality . The same metrics should be used tat the end of reengineering to identify the quality of the target system.
2. Once the current system characteristic, specification, quality and value is determined , we have to identify the specification and value of the target system such as hardware, software, operating systems, language and design.
3. Then create the set of test standards to explain the functionality of the new system is equivalent of the old system.
4. **Implement Reengineering:**

Once the analysis of the current system is done , the implementation of the new system starts. This involves two steps

1. **Reverse Engineering:**
2. **Forward engineering:**
3. **Conversion and Testing:**

Once the implementation is done , testing needs to be done to detect the errors. The same test cases should executed on legacy and new system and results to be compared to make sure the functionality is achieved. Test results should be douemented.

1. **What are non-functional requirements?**

**Functional Requirements: -** It describes what the system should do. It specifies the function and behavior of the system or component. It is described as documents or use cases. Use cases are the diagrammatic representation of the system behavior. It mostly covers the following,

1. Calculations
2. Technical Details
3. Data manipulation
4. Data processing

**Non Functional Requirements: -** It specifies how the system do. It specifies how the system should behave. It is used to judge the systems operation rather than specific behaviour. It describes the performance characteristic of the system. It is not the straight forward requirement of the system.

The non-functional requirements fall into the following areas.

1. Security
2. Scalability
3. Maintainability
4. Reliability
5. Efficiency
6. Stability
7. Testability
8. Supportability
9. Accessibility
10. Extensibility
11. Portability
12. Interoperability
13. **What is software development methodlogy ? How do you deliver your project ? What is the Agile methodology ?**

Software development methodology is a framework for structure , plan and control the software development process.

There are so many software development methodologies available, the most frequently used are,

1. Agile Methodology
2. Scrum Methodology
3. Waterfall Methodology
4. **What are the sections available in the technical design document?**