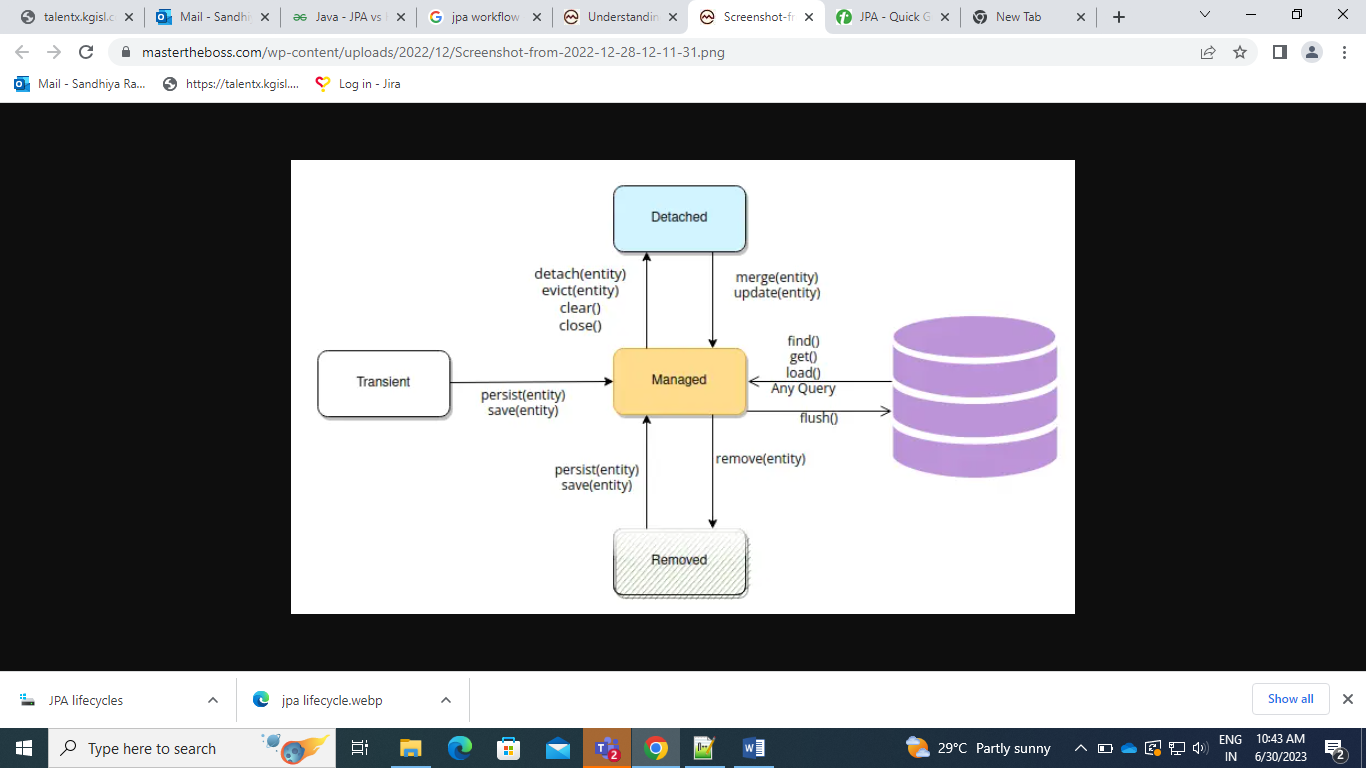
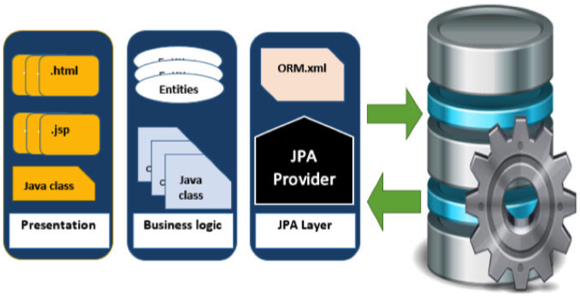
JAVA PERSISTENCE API (JPA)

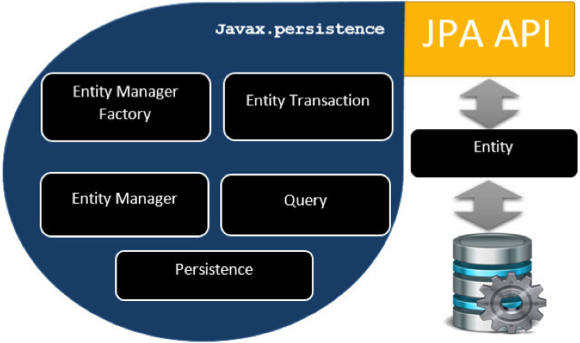
* JPA:
  + JPA is a collection of classes and methods to persistently store the vast amounts of data into a database which is provided by the Oracle Corporation.
* JPA LIFECYCLE:
  + The**JPA Entity Lifecycle**refers to the stages that an entity (a Java object representing a database record) goes through during its existence in a Java Persistence API (JPA) application.
  + There are four Main stages:
    - Transient
    - Managed
    - Detached
    - Removed
  + **TRANSIENT:**
    - When Entity is first created, it is in “New” state.
    - So, it is not associated with any persistence context and no representation of database.
  + **MANAGED:**
    - When Entity is associated with persistence context (EntityManager.persist() method), it becomes managed.
    - Any changes happens in entity will be persisted to database when transaction is committed.
* **DETACHED:**
  + If an Entity is removed from persistence context (EntityManager.detached() method), it becomes detached.
  + It no longer managed and any changes made will not be persisted to database.
* **REMOVED:**
  + If an Entity is removed from the database (EntityManager.remove () method), it becomes removed.
  + It no longer managed and any changes made will not be persisted to database.

****

* **JPA USE:**
  + To reduce the burden of writing codes for relational object management, a programmer follows the ‘JPA Provider’ framework, which allows easy interaction with database instance.

****

* **CLASS LEVEL ARCHITECTURE:**

****

* **ENTITY MANAGER FACTORY:**
  + It is a factory class of Entity Manager.
  + It creates and manages multiple Entity Manager Instances.
* **ENTITY MANAGER:**
  + It is an interface, it manages the persistent operations on objects.
  + It works like factory for query instances.
* **ENTITY:**
  + Entities are persistence objects, stores as records in database.
* **SPRING DATA JPA QUERY**
  + **@Query**
  + **@Param**
  + **@index Param**
  + **@Native Query**
* **PERSISTENCE:**
  + Persistence means **Storage.**
  + The place where we will store the data is called Persistence store.
* **PERSISTENT OPERATIONS:**
  + Inserting data
  + Updating data
  + Deleting data
  + Retrieve data
* **PERSISTENT TECHNOLOGIES:**
  + Several technologies to develop persistent logic
    - Java JDBC
    - Spring JDBC
    - ORM frameworks
    - Spring Data etc…
* **JPA Entity ASSOCIATION MAPPING**
  + @OneToOne - unidirectional
  + @OneToOne - bidirectional
  + @OneToMany – unidirectional
  + @OneToMany or @ManyToOne – bidirectional (both are same)
  + @ManyToOne - unidirectional
  + @ManyToMany – unidirectional
  + @ManyToMany - bidirectional
* **UNIDIRECTIONAL ASSOCIATION MAPPING:**
  + Unidirectional association mapping involve a relationship from one entity to another.
  + But the target entity (child class) does not have direct reference back to the source entity (Parent class).
  + **Parent class** - We only use @OneToOne or @OneToMany or @ManyToOne or @ManyToMany and @joinColumn (To join the foreign key column in parent table) in source entity (Parent class).
  + **EXAMPLE:** 
    - @ use @OneToOne or @OneToMany or @ManyToOne
    - **Employee** – Parent class, **Address** – Child class
* *@OneToMany*(cascade = *CascadeType*.***ALL***)
* *@JoinColumn*(name="foreign\_key\_address\_id")
* private List<Address> address;
  + - @ManyToMany – We use join table because it involves with multiple table relation.
    - **Book** – Parent class, **Author** – Child class
* *@ManyToMany*(cascade=*CascadeType*.***ALL***)
* *@JoinTable*(
* name="author\_book\_ManyToMany\_uni",
* joinColumns = *@JoinColumn*(name="book\_id"),
* inverseJoinColumns = *@JoinColumn*(name="author\_id")
* )
* private List<Author> authors;
* **BIDIRECTIONAL ASSOCIATION MAPPING:**
  + Bidirectional association mapping involves relationships between two entities, where each entity has a reference to the other.
  + **Parent class -** We use @OneToOne or @OneToMany or @ManyToOne or @ManyToMany and @joinColumn (To join the foreign key column in parent table) in source entity (Parent class).
  + **Child class –** For bidirectional relationships, we use @OneToOne or @OneToMany or @ManyToOne or @ManyToMany with (**mappedBy** = “child class field which referred in parent class”)
  + **EXAMPLE:**
    - Parent Class – Account.java
* *@ManyToOne*(cascade = *CascadeType*.***ALL***)
* *@JoinColumn*(name = "branch\_id")
* Branch branch;
  + - Child class – Branch.java
* *@OneToMany*(mappedBy = "branch", cascade = *CascadeType*.***ALL***)
* List<Account> account;
* **JPA ANNOTATIONS:**
  + **@Entity**
    - It is used to mark a java class as an entity, representing a table in the database.
  + **@Table**
    - Specifies the name of the database table, to which the entity is mapped.
  + **@Id**
    - Marks the field as primary key of the entity
  + **@GeneratedValue**
    - Used with the combination of **@Id**, it specifies how primary key value is generated.
    - STRATEGY TYPES:
      * **GenerationType.IDENTITY** – for auto-incrementing values.
      * **GenerationType.SEQUENCE –** for database sequence-based generation.
      * **GenerationType.AUTO-** let JPA choose appropriate strategy based on database.
      * **GenerationType.TABLE**
  + **@Column**
    - Used to map a class field to database column.
  + **@OneToOne**
  + **@OneToMany**
  + **@ManyToOne**
  + **@ManyToMany**
  + **@JoinColumn**
    - Used in relationship annotations.
    - It specifies the column that will be used as the join column in database.
  + **@Transient**
    - It marks the field should not be persisted (Continue) to the database.
    - It is used for temporary fields or other fields
  + **@NamedQuery and @NamedQueries**