TECHCAREER – FULLSTACK BOOTCAMP
TO-DO PROJESİ

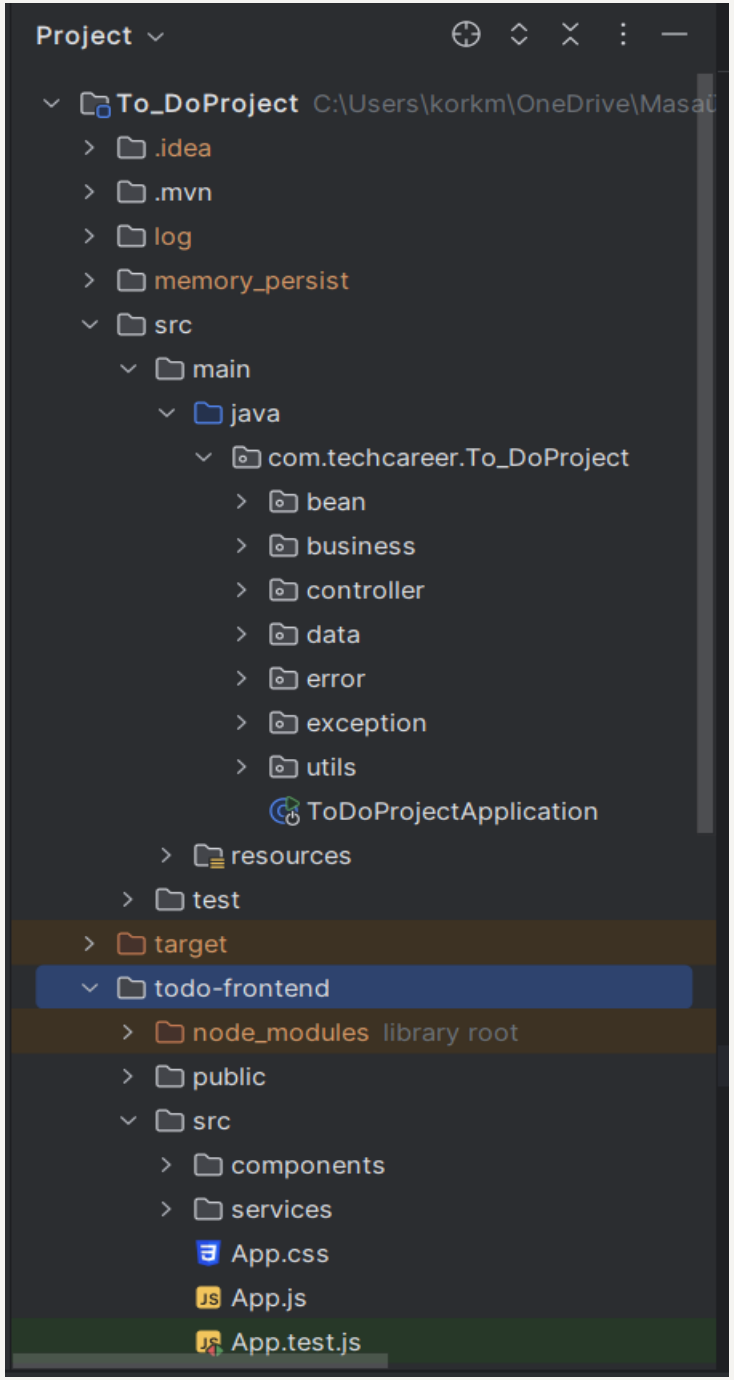
NAZAN KORKMAZ

20.05.2024

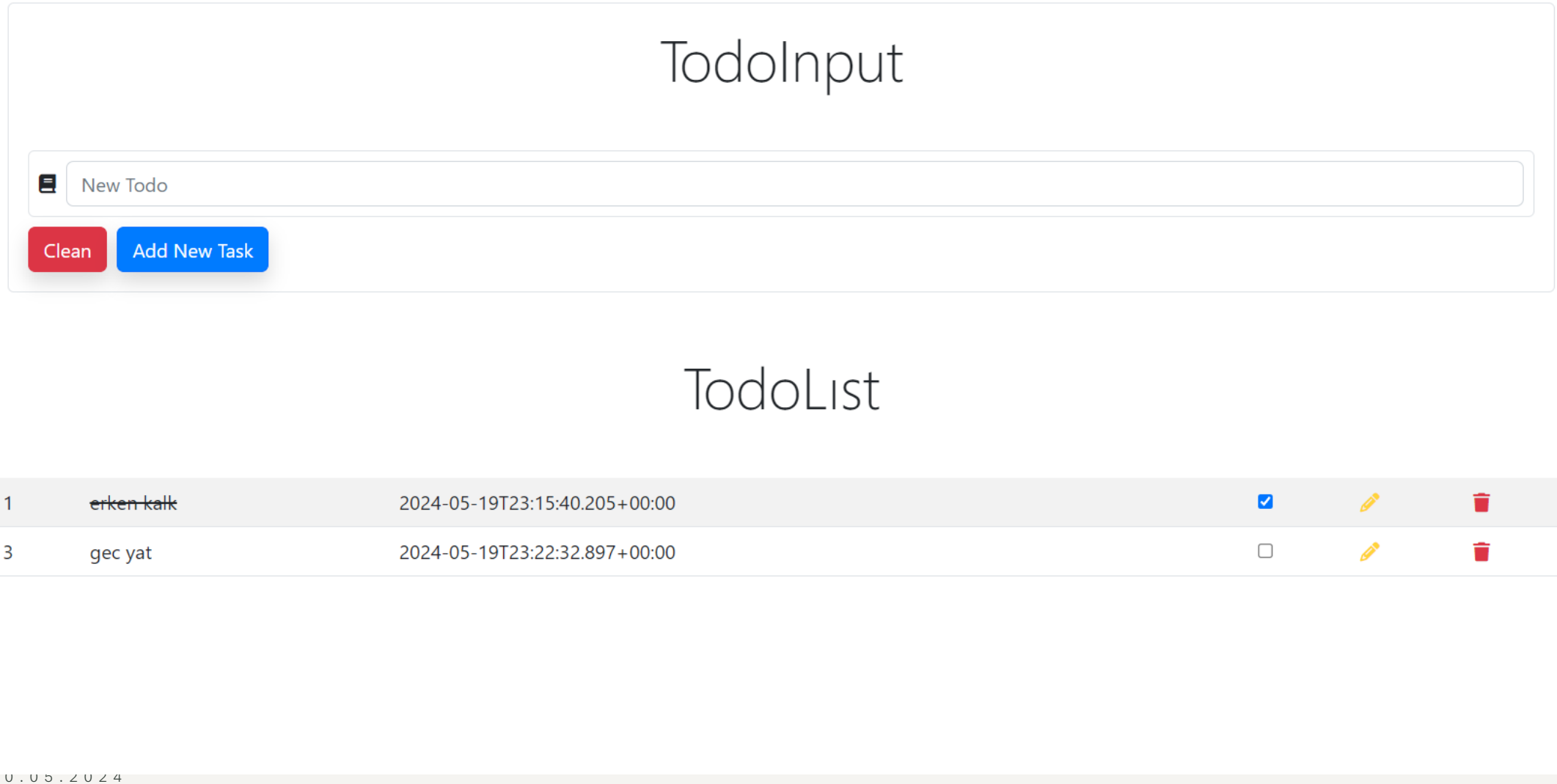
İÇİNDEKİLER

PROJE HAKKINDA

- To-Do uygulaması yapacaklarınızı listeleyebileceğiniz, bunları güncelleyebileceğiniz, silebileceğiniz ve ekleyebileceğiniz basit bir arayüzü olan kullanışlı bir uygulamadır.
  Görevlerinizi kolaylıkla takip etmenize yardımcı olacaktır.

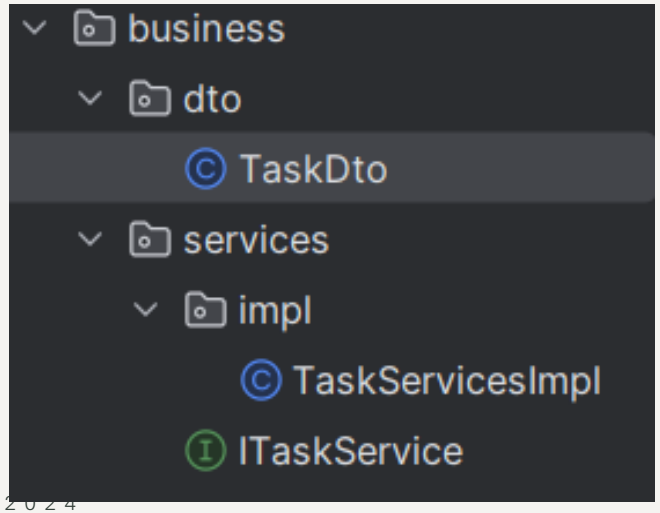


PROJE HAKKINDA



BUSSİNESS YAPISI

- Lombok kütüphanesinin özelliklerinden faydalanılarak bir DTO (Data Transfer Object) sınıfı olan TaskDto tanımlanmış. TaskDto sınıfı, bir görevi temsil etmek için kullanılan veri aktarım nesnesidir. Bu nesne, görevin kimlik numarası (taskId), adı (taskName), tamamlanma durumu (completed) ve oluşturulma tarihi (systemCreatedDate) gibi özelliklerini içerir. Bu özelliklerin bir kısmı, geçerlilik kuralları (@NotEmpty, @Size) ile doğrulanır ve bu, boş veya belirli bir boyutta olmayan değerlerin kabul edilmemesini sağlar.




```java
// LOMBOK
@Data                  //burada getter setterlar hashcode toString var  39 usages  Nazan Korkmaz
@AllArgsConstructor   //Parametreli constructorler silinir
@NoArgsConstructor    //parametresiz constructor silinir
@Log4j2              //loglama için cağırılır
@Builder             //design pattern yapısı için


public class TaskDto implements Serializable {

    // Serilestirme icin
    public static final Long serialVersionUID = 1L;

    // Task Id
    private Long taskId;

    // Task Name
    // Validation    ----> burada reactta bos gecmeye calıstığı zaman gecmezsin diye hata vericek
    @NotEmpty(message = "bos gorev olamaz!")
    @Size(min = 2, max = 100, message = "Gorev 2-100 karakter olmalı!")
    private String taskName;


    private boolean completed;

    private Date systemCreatedDate;
```

BUSSİNESS YAPISI

- Bu sınıf, bir servis hizmeti olarak tasarlanmış ve ITaskService arayüzünü uygulamaktadır. Bu arayüz, görevlerin oluşturulması, listelenmesi, güncellenmesi, silinmesi ve tamamlanma durumunun güncellenmesi gibi temel işlemleri tanımlar.veritabanı işlemlerini gerçekleştirir. Bu yöntemler, DTO ve Entity arasında dönüşümleri gerçekleştirir ve veritabanı işlemlerini yürütür. Örneğin, taskServiceCreate metodu bir görev oluştururken, taskServiceList metodu tüm görevleri listeler. updateTaskCompletionStatus metodu, bir görevin tamamlanma durumunu günceller.

```java
// Lombok
@RequiredArgsConstructor  Nazan Korkmaz *
@Log4j2
@Service
@Component("taskServicesImpl")   //spring tarafından springin bir parçasısın artık demek
public class TaskServicesImpl implements ITaskService<TaskDto, TaskEntity> {

    @Autowired
    private final TaskRepository taskRepository;
    private final ModelMapperBeanClass modelMapperBeanClass;

    @Override  4 usages  Nazan Korkmaz
    public TaskDto entityToDto(TaskEntity taskEntity) {
        return modelMapperBeanClass.modelMapperMethod().map(taskEntity, TaskDto.class);
    }

    @Override  1 usage  Nazan Korkmaz
    public TaskEntity dtoToEntity(TaskDto taskDto) {
        return modelMapperBeanClass.modelMapperMethod().map(taskDto, TaskEntity.class);
    }

    @Override        // database'e veri eklerken bu methot çağırılır  1 usage  Nazan Korkmaz *
    @Transactional  // Create,Update,Delete
    public TaskDto taskServiceCreate(TaskDto taskDto) {
        TaskEntity roleEntity1;
        // Dto => Entity çevirmek
        roleEntity1 = dtoToEntity(taskDto);
        roleEntity1.setTaskName(roleEntity1.getTaskName());
        // Kaydetmek
        TaskEntity roleEntity2 = taskRepository.save(roleEntity1);
        // ID ve Date Dto üzerinde Set yapıyorum
```

BUSSİNESS YAPISI

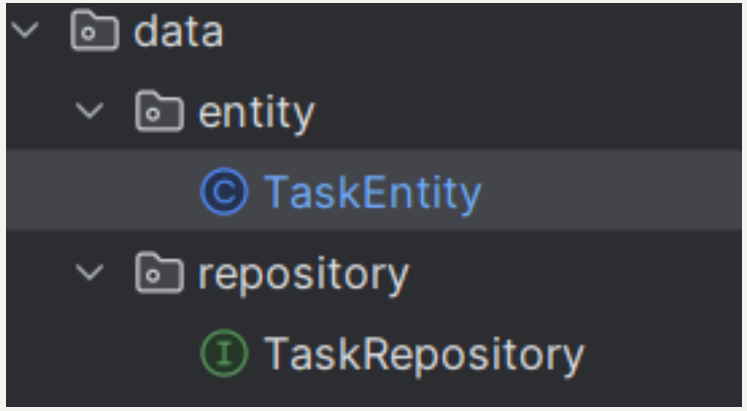
- Bu arayüz, DTO ve Entity sınıfları arasında veri dönüşümlerini gerçekleştiren ve temel iş mantığı operasyonlarını tanımlayan bir interface sağlar.
- entityToDto(E e): Bir Entity nesnesini alır ve onu ilgili DTO nesnesine dönüştürür. dtoToEntity(D d): Bir DTO nesnesini alır ve onu ilgili Entity nesnesine dönüştürür.
- taskServiceCreate(D d): Verilen DTO nesnesini kullanarak bir Entity oluşturur ve veritabanına kaydeder. Oluşturulan görevin DTO temsilini döndürür.
- taskServiceList(): Tüm görevleri listelemek için kullanılır. Görevleri alır, bunları DTO nesnelerine dönüştürür ve bir liste olarak döndürür.
- taskServiceFindById(Long id): Verilen ID'ye sahip görevi veritabanından bulur, bunu bir DTO nesnesine dönüştürür ve döndürür.
- taskServiceUpdateById(Long id, D d):Verilen ID'ye sahip görevi bulur, verilen DTO nesnesini kullanarak günceller ve güncellenmiş görevin DTO temsilini döndürür.

```
// D: Dto
// E: Entity
public interface ITaskService <D, E>{  4 usages  1 implementation  Nazan Korkmaz *
    // MODEL MAPPER
    public D entityToDto(E e);  4 usages  1 implementation  Nazan Korkmaz
    public E dtoToEntity(D d);  1 usage  1 implementation  Nazan Korkmaz

    // TASK CRUD
    public D taskServiceCreate(D d);  1 usage  1 implementation  Nazan Korkmaz

    public List<D> taskServiceList();  1 usage  1 implementation  Nazan Korkmaz

    public D taskServiceFindById(Long id);  2 usages  1 implementation  Nazan Korkmaz

    public D taskServiceUpdateById(Long id, D d);  1 usage  1 implementation  Nazan Kor

    public void taskServiceDeleteById(Long id);  1 usage  1 implementation  Nazan Korkm

    public D updateTaskCompletionStatus(Long id, Boolean completed, D d);  1 usage
}
```

- taskServiceDeleteById(Long id): Verilen ID'ye sahip görevi bulur ve veritabanından siler.
- updateTaskCompletionStatus(Long id, Boolean completed, D d): Verilen ID'ye sahip görevi bulur, tamamlanma durumunu günceller ve güncellenmiş görevin DTO temsilini döndürür.

DATA YAPISI

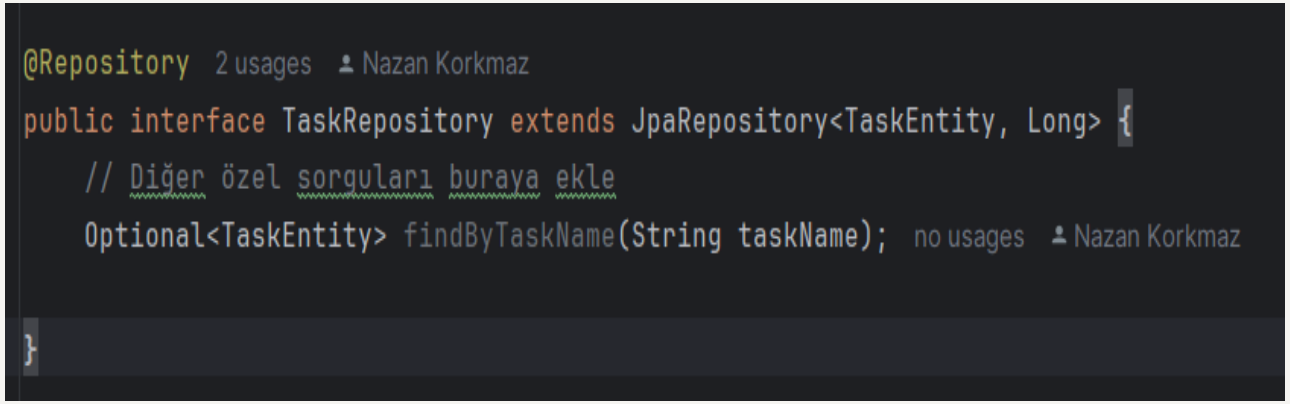
- Veritabanındaki görevleri temsil etmek için kullanılır.
- @Entity(name = "Tasks"): Bu sınıfın bir JPA varlık sınıfı olduğunu belirtir ve veritabanındaki tablonun adını "Tasks" olarak belirler.
- @Table(name = "tasks"): Veritabanındaki gerçek tablonun adını belirtir. Bu durumda, "tasks" tablosuyla eşleşir.
- Yapılandırıcılar ve diğer standart yöntemler: Sınıfa eklenen diğer yapılandırıcılar ve yöntemler, sınıfın oluşturulması ve kullanımı için standart işlevselliği sağlar.




```java
//ENTITY  ----> bunun database'e verileri eklmesi için çalışıcam
@Entity(name = "Tasks")
@Table(name = "tasks")

public class TaskEntity implements Serializable {

    public static final Long serialVersionUID = 1L;
    // Task Id
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)  // id'nin artarak devam etmesini sağlar
    @Column(name = "task_id") //database'e bu adla kaydetsin
    private Long taskId;

    // Task Name
    @Column(name = "task_name")
    private String taskName;

    // Task Completed
    @Column(name = "completed")
    private boolean completed;

    //System Created Date
    @CreationTimestamp
    @Temporal(TemporalType.TIMESTAMP) //zaman sn dk falan versin
    private Date systemCreatedDate;

    public TaskEntity(Long taskId, String taskName, boolean completed, Date systemCreatedDate) {...}

    public boolean isCompleted() {  no usages  Nazan Korkmaz
        return completed;
    }
    public void setCompleted(boolean completed) {  2 usages  Nazan Korkmaz
```

DATA YAPISI

- JPA (Java Persistence API) tarafından sağlanan JpaRepository arayüzünü genişleten bir arayüzdür.
- @Repository: Bu sınıfın bir Spring bileşeni olduğunu belirtir ve Spring tarafından otomatik olarak bulunup yönetilmesini sağlar.
- public interface TaskRepository extends JpaRepository<TaskEntity, Long>: TaskEntity sınıfı için bir JPA repository'si tanımlar. JpaRepository, JPA varlıklarını veritabanı işlemleri için temel CRUD (Create, Read, Update, Delete) işlemlerini sağlar. TaskEntity, veritabanında görevlerle ilgili verilere erişmek için kullanılacak varlık sınıfıdır. Long, bu varlık sınıfının birincil anahtarının türünü belirtir.
- Optional<TaskEntity> findByTaskName(String taskName): Bu metot, görev adı kullanılarak belirli bir görevi veritabanından bulmak için bir özel sorgu sağlar. Bu metot, TaskEntity sınıfında tanımlanan taskName alanını kullanarak bir görevi arar ve bulunursa Optional bir TaskEntity nesnesi döndürür. Bu, özel bir sorgudur.

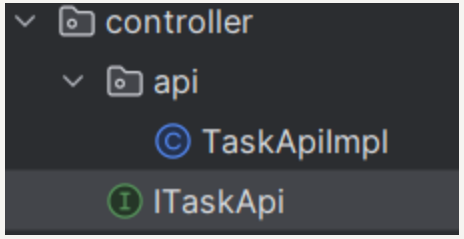


CONTROLLER YAPISI

- public ResponseEntity<?> taskApiCreate(D d): Yeni bir görev oluşturmak için kullanılacak metodun imzasıdır. Bu metod, istemci tarafından sunulan görev verisini alır ve veritabanına ekler. Sonuç olarak, oluşturulan görevin bilgilerini içeren ResponseEntity döndürür.
- public ResponseEntity<List<D>> taskApiList(): Tüm görevleri listeleyen metodun imzasıdır. ResponseEntity<List<D>> türünde bir cevap döndürür.
- public ResponseEntity<?> taskApiFindById(Long id): Bu metod, verilen bir görev kimliği (id) ile ilgili görevi veritabanından bulur ve ResponseEntity içinde görev bilgilerini döndürür.
- public ResponseEntity<?> taskApiUpdateById(Long id, D d): Bu metod, verilen bir görev kimliği (id) ile ilgili görevi ve güncellenmiş görev verisini alır, veritabanında ilgili görevi günceller ve bir ResponseEntity içinde güncellenmiş görev bilgilerini döndürür.

```
public interface ITaskApi <D>{  2 usages  1 implementation  Nazan Korkmaz *
    // TASK CRUD

    public ResponseEntity<?> taskApiCreate(D d);  no usages  1 implementation  Nazan Korkmaz

    public ResponseEntity<List<D>> taskApiList();  no usages  1 implementation  Nazan Korkmaz

    public ResponseEntity<?> taskApiFindById(Long id);  no usages  1 implementation  Nazan Korkmaz

    public ResponseEntity<?> taskApiUpdateById(Long id, D d);  no usages  1 implementation  Nazan

    public ResponseEntity<String> taskApiDeleteById(Long id);  no usages  1 implementation  Nazan

    public ResponseEntity<?> updateTaskCompletionStatus(Long id,boolean isCompleted, D d );
}
```

- public ResponseEntity<String> taskApiDeleteById(Long id): Bu metod, verilen bir görev kimliği (id) ile ilgili görevi veritabanından siler ve bir ResponseEntity içinde bir mesaj döndürür.
- public ResponseEntity<?> updateTaskCompletionStatus(Long id,boolean isCompleted, D d): Bu metod, verilen bir görev kimliği (id) ile ilgili görevin tamamlanma durumunu günceller ve bir ResponseEntity içinde güncellenmiş görev bilgilerini döndürür.



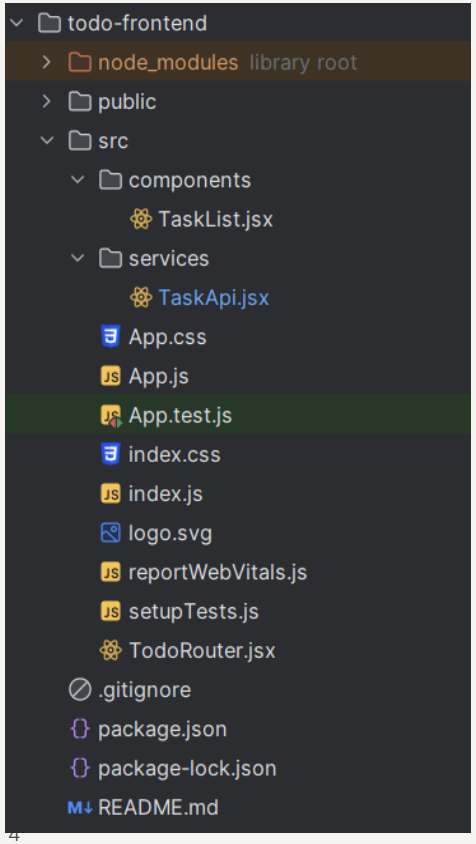
CONTROLLER YAPISI

- Task işlemlerini gerçekleştiren REST API'nin uygulama katmanını oluşturur. Yapılmak istenenler şunlardır:
- @RestController ve @RequestMapping("/api/v1.0.0") anotasyonları, bu sınıfın bir REST denetleyicisi olduğunu ve tüm isteklerin "/api/v1.0.0" yolundan işleneceğini belirtir.
- @PostMapping("/create"), @GetMapping("/list"), @GetMapping({"/find","/find/{id}"}), @PutMapping({"/update","/update/{id}"}), @DeleteMapping({"/delete","/delete/{id}"}), @PutMapping("/update/completion/{id}") gibi HTTP metodları için uygun mapping'leri sağlar. Bu annotation'lar, ilgili HTTP metoduna göre gelen istekleri işleyen metodları belirler.
- taskApiCreate, taskApiList, taskApiFindById, taskApiUpdateById, taskApiDeleteById, updateTaskCompletionStatus gibi metodlar, belirli işlemleri gerçekleştiren REST API endpoint'lerini tanımlar. Örneğin, taskApiCreate metodu yeni bir görev oluşturmak için kullanılırken, taskApiUpdateById metodu belirli bir görevi güncellemek için kullanılır.
- @CrossOrigin anotasyonunu CORS hatasını engeller ve farklı bir kaynaktan gelen isteklere izin verir.

```java
@RequestMapping("/api/v1.0.0")
@CrossOrigin //CORS: Hatası
public class TaskApiImpl implements ITaskApi<TaskDto> {
    // Injection
    @Qualifier("taskServicesImpl")
    private final ITaskService iTaskService;
    // Error
    private ApiResult apiResult;  no usages

    // CREATE (Api)
    // http://localhost:4444/api/v1.0.0/create
    @PostMapping("/create")  Nazan Korkmaz *
    @Override
    public ResponseEntity<?> taskApiCreate(@RequestBody TaskDto taskDto) {
        TaskDto taskCreateApi =(TaskDto) iTaskService.taskServiceCreate(taskDto);
        if (taskCreateApi == null) {...} else if (taskCreateApi.getTaskId() == 0) {
            ApiResult apiResultCreate = ApiResult.builder()
                    .status(400)
                    .error("Task Eklenmedi")
                    .message("Task Dto Bad Request")
                    .path("localhost:4444/api/v1.0.0/create")
                    .createdDate(new Date(System.currentTimeMillis()))
                    .build();
            return ResponseEntity.status(400).body(apiResultCreate);
        }
        log.info("Task Api eklendi");
        return ResponseEntity.status(201).body(taskCreateApi);
    }

    @GetMapping("/list")  Nazan Korkmaz
    @Override
    public ResponseEntity<List<TaskDto>> taskApiList() {
```

SERVİCES YAPISI

- TodoRouter, uygulamanın yönlendirme mantığını oluşturur. Farklı URL'lere göre ilgili bileşenlerin render edilmesini sağlar.
- react-router-dom kütüphanesinden Route ve Routes bileşenleri kullanılarak farklı URL'ler için yönlendirme yapılandırılır.




```jsx
import React from 'react'
import { Route, Routes } from 'react-router-dom'
import { Navigate } from 'react-router-dom';
import TaskList from "./components/TaskList";

function TodoRouter() {  Show usages  Nazan Korkmaz *
  return (
    <React.Fragment>
        <div className="container">
            <Routes>
            <Route path={"/"} element={<TaskList/>} />
            <Route path={"/index"} element={<TaskList/>} />
                {/* List */}
                        <Route path={"/list"} element={<TaskList/>} />
                        <Route path={"*"} element={<Navigate to={"/"} />} />
            </Routes>
        </div>
    </React.Fragment>
    //<div>TodoRouter</div>
  )
}
export default TodoRouter;  Show usages  Nazan Korkmaz
```

- TaskList bileşeni, /, /index ve /list URL'lerine atanır. Bu sayede, bu URL'lere yapılan isteklerde TaskList bileşeninin render edilmesi sağlanır.
-
- <Navigate to={"/"} /> bileşeni, belirli olmayan URL'ler için ana sayfaya yönlendirme yapar. Yani, / ve /index dışındaki herhangi bir URL'ye gidildiğinde ana sayfaya (/) yönlendirilir.

SERVİCES YAPISI

- CRUD (Create, Read, Update, Delete) işlemlerini gerçekleştirmek için gerekli HTTP isteklerini yapar.
- taskApiCreate(taskDto): Yeni bir görev oluşturmak için POST isteği yapar.
- taskApiList(): Tüm görevleri listelemek için GET isteği yapar.
- taskApiFindById(id): Belirli bir görevi bulmak için GET isteği yapar.
- taskApiUpdateById(id, taskDto): Belirli bir görevi güncellemek için PUT isteği yapar.
- taskApiDeleteById(id): Belirli bir görevi silmek için DELETE isteği yapar.
- taskApiUpdateCompletionStatus(taskId, completed, taskDto): Görevin tamamlanma durumunu güncellemek için PUT isteği yapar.
- HTTP isteklerini yönetmek için kullanılır ve uygulamanın iş mantığına uygun şekilde API isteklerini gerçekleştirir.

```
import axios from 'axios';

const TODO_API_PERSIST_URL : string ="/api/v1.0.0"
class TaskApi{ Show usages Nazan Korkmaz *
    // CREATE (Api)...
    taskApiCreate(taskDto) : any { Show usages Nazan Korkmaz
        return axios.post( url: `${TODO_API_PERSIST_URL}/create`,taskDto);
    }

    //@GetMapping("/list")
    taskApiList() : any { Show usages Nazan Korkmaz
        return axios.get( url: `${TODO_API_PERSIST_URL}/list`);
    }

    //@GetMapping({"/find","/find/{id}"})
    taskApiFindById(id) : any { no usages new *
        return axios.get( url: `${TODO_API_PERSIST_URL}/find/${id}`);}


    //@PutMapping({"/update","/update/{id}"})
    taskApiUpdateById(id, taskDto) : any { Show usages Nazan Korkmaz
        return axios.put( url: `${TODO_API_PERSIST_URL}/update/${id}`,taskDto);
    }

    //@DeleteMapping({"/delete","/delete/{id}"})
    taskApiDeleteById(id) : any { Show usages Nazan Korkmaz
        return axios.delete( url: `${TODO_API_PERSIST_URL}/delete/${id}`);
    }

    taskApiUpdateCompletionStatus(taskId, completed,taskDto) : any { no usages Nazan Korkmaz
taskApiList()
```

COMPONENT YAPISI

- Kullanıcı arayüzü üzerinde görevleri listeler ve yönetir.
- useEffect hook'u kullanılarak bileşen oluşturulduğunda, fetchTaskList fonksiyonu çağrılır ve görev listesi alınır.
- fetchTaskList fonksiyonu, TaskApi üzerindeki taskApiList metodunu kullanarak görev listesini getirir ve bu listeyi bileşenin durumuna (taskListData state'ine) kaydeder.
- setUpdateTask fonksiyonu, bir görevin güncellenmesi için modal penceresini açar. Bu fonksiyon, seçilen görevi alır, modal penceresini gösterir ve kullanıcının görev adını düzenlemesine izin verir.
- handleClose fonksiyonu, modal penceresini kapatır ve bileşenin durumunu temizler.
- handleSaveChanges fonksiyonu, modal penceresindeki düzenlenmiş görevin kaydedilmesi için çağrılır. Güncellenmiş görev TaskApi üzerindeki taskApiUpdateById metodu kullanılarak güncellenir ve güncellenmiş görev listesi alınır.

```
import ...


function TaskList() {  Show usages  Nazan Korkmaz *
    const navigate : NavigateFunction  = useNavigate();
    const [taskListData : any[] , setTaskListData] = useState( initialState: []);
    const [taskName : string , setTaskName] = useState( initialState: '');
    const [error, setError] = useState( initialState: undefined);
    const [spinner : boolean , setSpinner] = useState( initialState: false);
    const [multipleRequest : boolean , setMultipleRequest] = useState( initialState: false);
    const [showModal : boolean , setShowModal] = useState( initialState: false);
    const [selectedTask, setSelectedTask] = useState( initialState: null);

    useEffect( effect: () : void  => {
        fetchTaskList();
    }, deps: []);
//
    const fetchTaskList = async () : Promise<void>  => {  Show usages  Nazan Korkmaz
        try {
            const response : AxiosResponse<any>  = await TaskApi.taskApiList();
            setTaskListData(response.data);
        } catch (error) {
            console.error('Error fetching tasks:', error);
        }
    };

    const setUpdateTask = (task) : void  => {  Show usages  Nazan Korkmaz
        setSelectedTask(task);
        setTaskName(task.taskName);
        setShowModal( value: true);
    };
```

COMPONENT YAPISI

- setDeleteTask fonksiyonu, belirli bir görevin silinmesini sağlar. Kullanıcıya bir onay iletişim kutusu gösterilir ve onaylanırsa, görev TaskApi üzerindeki taskApiDeleteById metodu kullanılarak silinir.
- toggleTaskCompletion fonksiyonu, bir görevin tamamlanma durumunu tersine çevirir. Bu, bir onay kutusu işaretlendiğinde veya işaret kaldırıldığında gerçekleşir.
- TaskCreateSubmit fonksiyonu, kullanıcının girdiği görev adı ve diğer bilgiler TaskApi üzerindeki taskApiCreate metodu kullanılarak gönderilir.
- Bileşen, bir tablo içinde görevleri listeler. Her bir görev için, görevin ID'si, adı, oluşturulma tarihi, tamamlanma durumu ve düzenle/sil butonları bulunur.
- Görevlerin tamamlanma durumu değiştirildiğinde, checkbox'ın durumu güncellenir ve toggleTaskCompletion fonksiyonu çağrılır.
- Modal penceresi, görev adının düzenlenebilmesi için kullanıcıya bir form sağlar. Kullanıcı düzenlemeyi kaydettiğinde, handleSaveChanges fonksiyonu çağrılır ve güncellenmiş görev TaskApi üzerinde güncellenir.

```
const setDeleteTask = async (taskId) : Promise<void>  => {  Show usages  Nazan K
    if (window.confirm(`${taskId} id datayı silmek istiyor musunuz ?`)) {
        try {...} catch (error) {
            console.error(error);
            navigate('/list');
        }
    } else {
        alert(`${taskId} nolu data silinmedi !!!`);
        window.location = "/list";
    }
};

const toggleTaskCompletion = async (taskId, completed) : Promise<void>  => {
    const task = taskListData.find(task => task.taskId === taskId);
    if (task) {...}
};

const clear = () : void  => {  Show usages  Nazan Korkmaz
    setTaskName( value: '');
};

const taskNameOnChange = (event) : void  => {  Show usages  Nazan Korkmaz
    const { value } = event.target;
    setTaskName(value);
};

const onSubmitForm = (event) : void  => {  Show usages  Nazan Korkmaz
    event.preventDefault();
};
```

DİNLEDİĞİNİZ İÇİN TEŞEKKÜR EDERİM