

Creating a ToDo list project using Spring Boot, Hibernate, PostgreSQL, and Docker involves several steps. Below is a comprehensive guide to setting up the project.

## Step 1: Set Up PostgreSQL with Docker

### 1. Create a Docker Compose File (docker-compose.yml):

```
yaml
Copy code
version: '3.1'

services:
  db:
    image: postgres:latest
    environment:
      POSTGRES_DB: todolist
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"
```

### 2. Run PostgreSQL Container:

```
sh
Copy code
docker-compose up -d
```

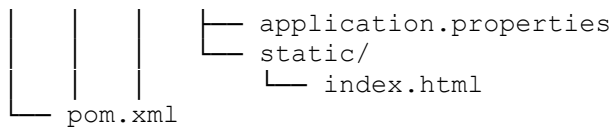
## Step 2: Create Spring Boot Project

### 1. Initialize the Spring Boot Project:

- Use [Spring Initializr](#) to generate the project with the following dependencies:
  - Spring Web
  - Spring Data JPA
  - PostgreSQL Driver
- Download the project and unzip it.

### 2. Project Structure:

```
css
Copy code
todolist/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── com/
│   │   │   │   ├── example/
│   │   │   │   │   ├── todolist/
│   │   │   │   │   │   ├── ToDoListApplication.java
│   │   │   │   │   │   ├── controller/
│   │   │   │   │   │   │   ├── ToDoController.java
│   │   │   │   │   │   ├── model/
│   │   │   │   │   │   │   ├── ToDoItem.java
│   │   │   │   │   │   ├── repository/
│   │   │   │   │   │   │   ├── ToDoRepository.java
│   │   │   │   │   │   ├── service/
│   │   │   │   │   │   │   ├── ToDoService.java
│   │   │   │   │   │   └──
│   │   │   │   │   └──
│   │   │   │   └──
│   │   │   └──
│   │   └── resources/
```



## Step 3: Configure Spring Boot Application

### 1. Update `application.properties`:

```
properties
Copy code
spring.datasource.url=jdbc:postgresql://localhost:5432/todolist
spring.datasource.username=postgres
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

### 2. Main Application (`ToDoListApplication.java`):

```
java
Copy code
package com.example.todolist;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ToDoListApplication {
    public static void main(String[] args) {
        SpringApplication.run(ToDoListApplication.class, args);
    }
}
```

## Step 4: Define Model, Repository, Service, and Controller

### 1. Model (`ToDoItem.java`):

```
java
Copy code
package com.example.todolist.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class ToDoItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String description;
    private boolean completed;

    // Getters and Setters
}
```

## 2. Repository (ToDoRepository.java):

```
java
Copy code
package com.example.todolist.repository;

import com.example.todolist.model.ToDoItem;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ToDoRepository extends JpaRepository<ToDoItem, Long>
{
}
```

## 3. Service (ToDoService.java):

```
java
Copy code
package com.example.todolist.service;

import com.example.todolist.model.ToDoItem;
import com.example.todolist.repository.ToDoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ToDoService {
    @Autowired
    private ToDoRepository toDoRepository;

    public List<ToDoItem> getAllToDoItems() {
        return toDoRepository.findAll();
    }

    public ToDoItem addToDoItem(ToDoItem toDoItem) {
        return toDoRepository.save(toDoItem);
    }

    public void deleteToDoItem(Long id) {
        toDoRepository.deleteById(id);
    }
}
```

## 4. Controller (ToDoController.java):

```
java
Copy code
package com.example.todolist.controller;

import com.example.todolist.model.ToDoItem;
import com.example.todolist.service.ToDoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/todo")
```

```

public class ToDoController {
    @Autowired
    private ToDoService toDoService;

    @GetMapping
    public List<ToDoItem> getAllToDoItems() {
        return toDoService.getAllToDoItems();
    }

    @PostMapping
    public ToDoItem addToDoItem(@RequestBody ToDoItem toDoItem) {
        return toDoService.addToDoItem(toDoItem);
    }

    @DeleteMapping("/{id}")
    public void deleteToDoItem(@PathVariable Long id) {
        toDoService.deleteToDoItem(id);
    }
}

```

## Step 5: Build and Run the Application

### 1. Build the Application:

```

sh
Copy code
mvn clean install

```

### 2. Run the Application:

```

sh
Copy code
mvn spring-boot:run

```

## Step 6: Test the Application

### 1. Access the API:

- Get all ToDo items: GET `http://localhost:8080/api/todo`
- Add a new ToDo item: POST `http://localhost:8080/api/todo`

```

json
Copy code
{
    "description": "Learn Docker",
    "completed": false
}

```

- Delete a ToDo item: DELETE `http://localhost:8080/api/todo/{id}`

This setup allows you to create a simple ToDo list application using Spring Boot, Hibernate, PostgreSQL, and Docker. The application can be further extended and customized based on your requirements.