

Design Principles

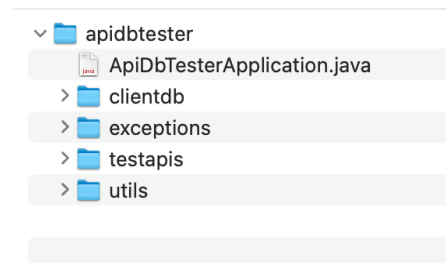
The project has implemented following design principles:

Modularity:

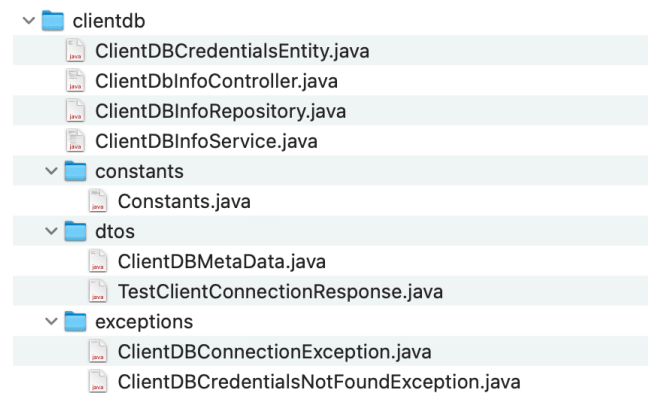
The code is highly modular. The backend is divided into 4 packages with each package handling related set of responsibilities. These packages have loose coupling among them and have less interaction with each other.

The classes are highly cohesive. All the modules have either stamp coupling i.e. one module uses only the fraction data structure passed by the other module or data coupling i.e. one module uses the whole data structure passed to it by other module. So, best coupling techniques have been implemented.

The backend package has 4 packages:



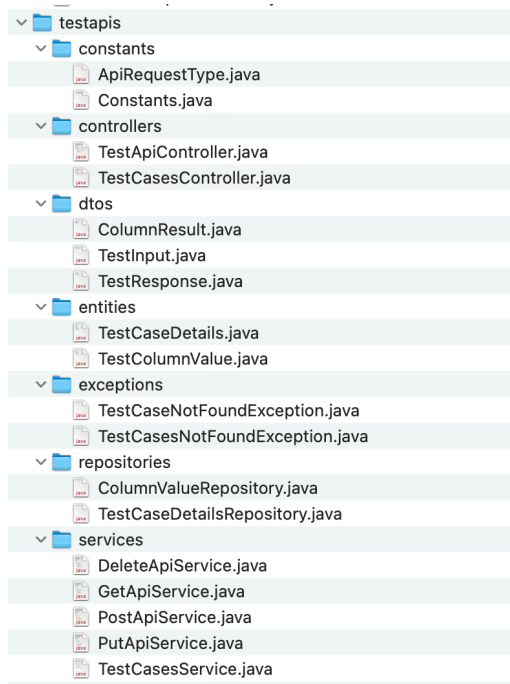
Clientdb package: It has functionality related to fetching information related to the client database application. It is further divided on basis of functionalities.



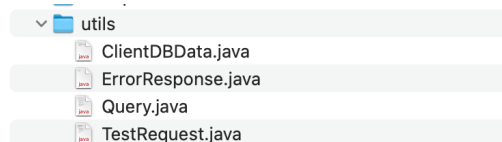
Exception package: It is used to handle the common exceptions that occur in the other packages.



Testapis package: It handles the functionality related to the testing of the client application's APIs.



Utils package: It handles the common functionality required by the different packages.



Encapsulation:

The encapsulation has been implemented. Only the data members and methods which are needed by other classes have been declared public. All others have been declared private. Example of a class is shown below:

```

import ...

6 usages  ± Vatsal Jain +1
@Service
public class PostApiService {

    @Autowired
    private TestCaseDetailsRepository testCaseDetailsRepository;

    @Autowired
    private ColumnValueRepository columnValueRepository;

    @Autowired
    private ClientDBInfoService clientDBInfoService;

    @Autowired
    private ModelMapper modelMapper;

    1 usage
    private TestRequest testRequest = new TestRequest();
    1 usage
    private ClientDBData clientDBData = new ClientDBData();

    ± Shubham Mishra +1
    public TestResponse fetchTestResult(TestInput testInput) {...}

    1 usage  ± Vatsal Jain +1
    private List<TestColumnValue> getTestColumnValuesWithResults(Connection connection, List<TestColumnValue> testColumnValues, TestCaseDetails testCaseDetails) {...}






    1 usage  ± Vatsal Jain +1
    private void saveTestColumnValues(List<TestColumnValue> testColumnValues, TestCaseDetails testCaseDetails) {...}

    1 usage  ± Shubham Mishra +1
    private boolean allTestPassed(List<TestColumnValue> testColumnValues) {...}
}

```

Single responsibility:

Each class has single responsibility. The classes have been made based on their functionalities and no class handles multiple responsibilities. Example of single responsibility is shown below:

-  DeleteApiService.java
-  GetApiService.java
-  PostApiService.java
-  PutApiService.java
-  TestCasesService.java

Repository pattern, dependency inversion principle and abstraction:

Repository interfaces have been created to access the database of the application. So, the domain and data mapping layers have been separated and the complexity involved in accessing database has been reduced making code more abstract. It also implements dependency inversion principle as classes needing data from database use interface to interact with it instead of concrete implementation. Example of repository pattern and abstraction is given below:

```
package com.project.apidbtester.testapis.repositories;
```

```
import ...
```

14 usages  Shubham Mishra

```
public interface ColumnValueRepository extends JpaRepository<TestColumnValue, Integer> {  
}
```