To Do List Project

Ali Hamza Mohammed

https://github.com/alihamzamohammed/IMS-Starter/

Intro

I decided to make a regular To Do list, to create and store items to do.

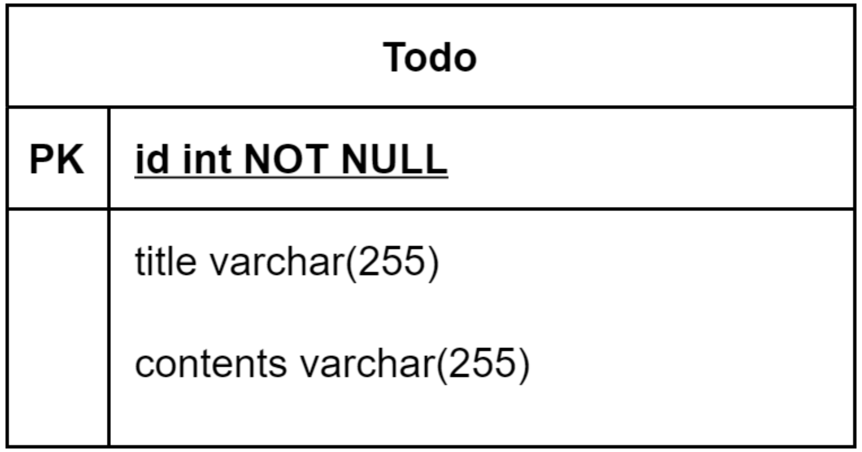
Originally, it was just to be To Do items, but halfway through I added categories to group these items.

Each category can have many items, and To Do items can have no categories, to which they belong to an auto generated ’Unsorted’ category.

Plan

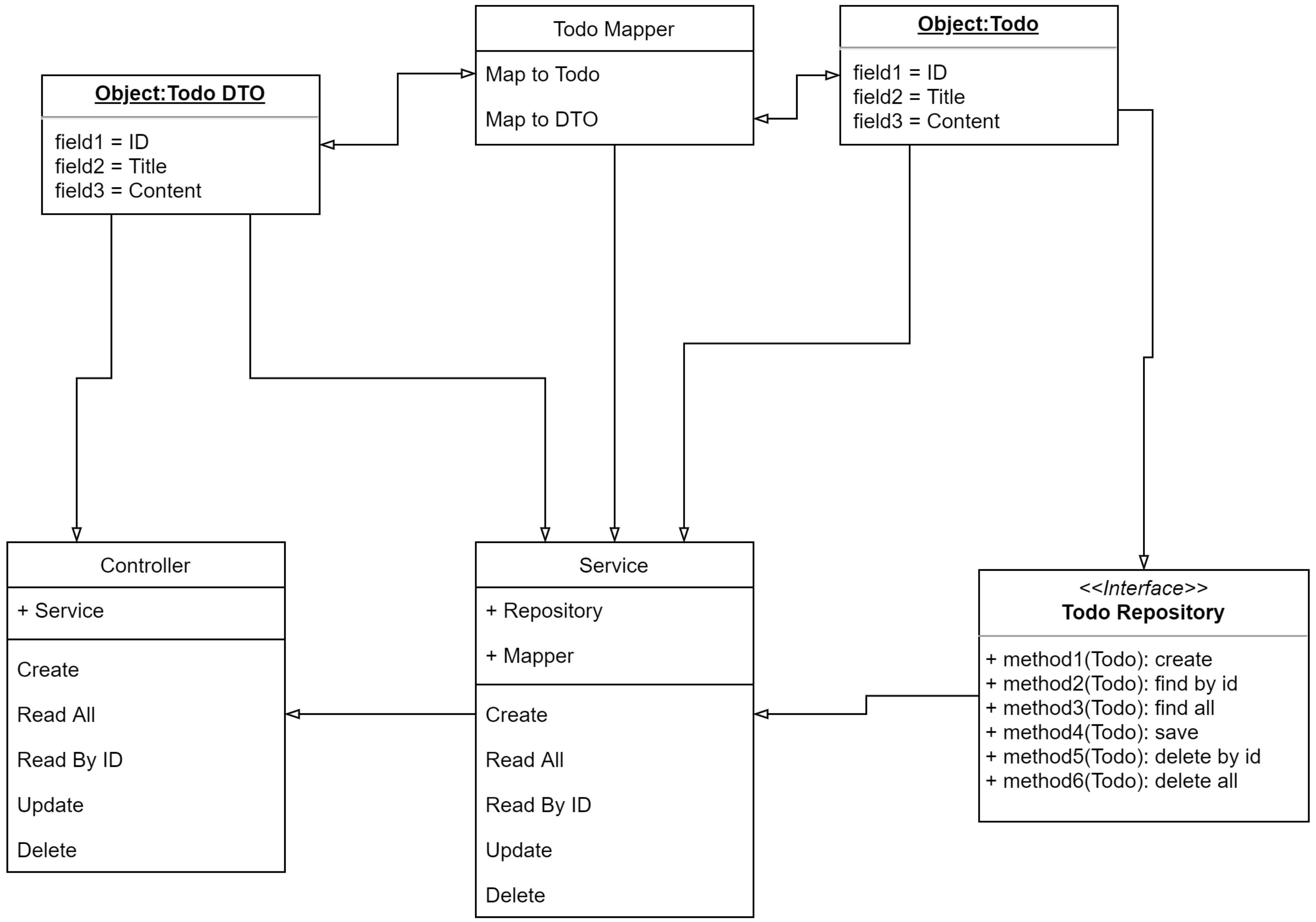
My original plan:

- One table, with ID, Title, and Contents, which represents all To Do items stored



Todo	
PK	**<u>id int NOT NULL</u>**
	title varchar(255) contents varchar(255)

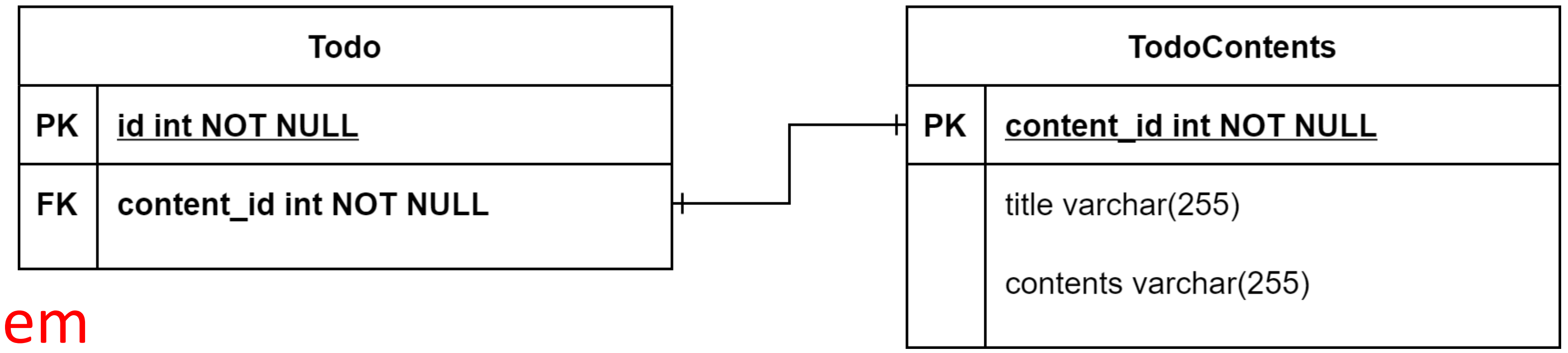
Problem: Had null records


Todo Mapper
Map to Todo
Map to DTO
Object:Todo
field1 = ID
field2 = Title
field3 = Content
Object:Todo DTO
field1 = ID
field2 = Title
field3 = Content
Controller
+ Service
Create
Read All
Read By ID
Update
Delete
Service
+ Repository
+ Mapper
Create
Read All
Read By ID
Update
Delete
<<Interface>>
Todo Repository
+ method1(Todo): create
+ method2(Todo): find by id
+ method3(Todo): find all
+ method4(Todo): save
+ method5(Todo): delete by id
+ method6(Todo): delete all

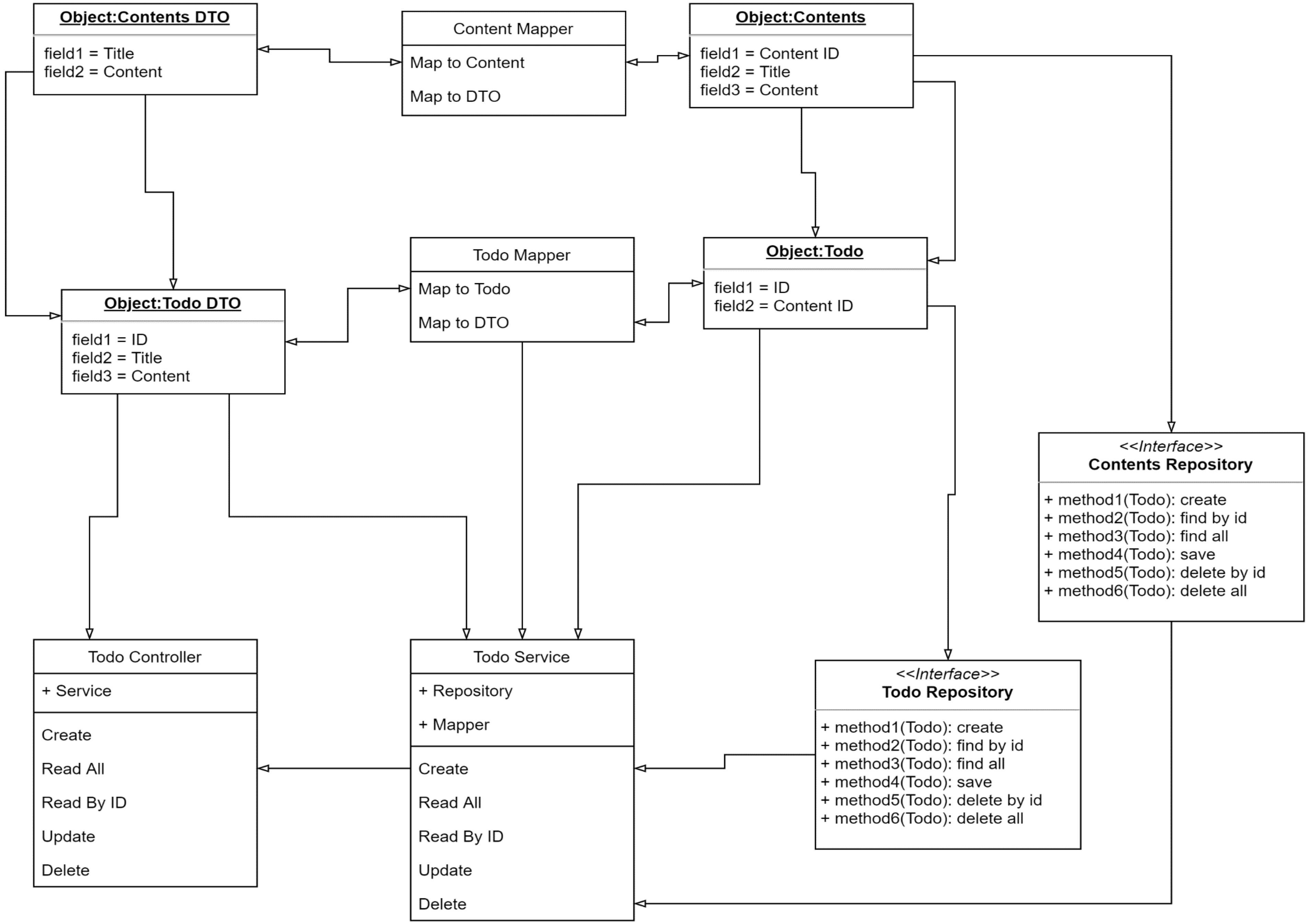
Plan

My new plan:

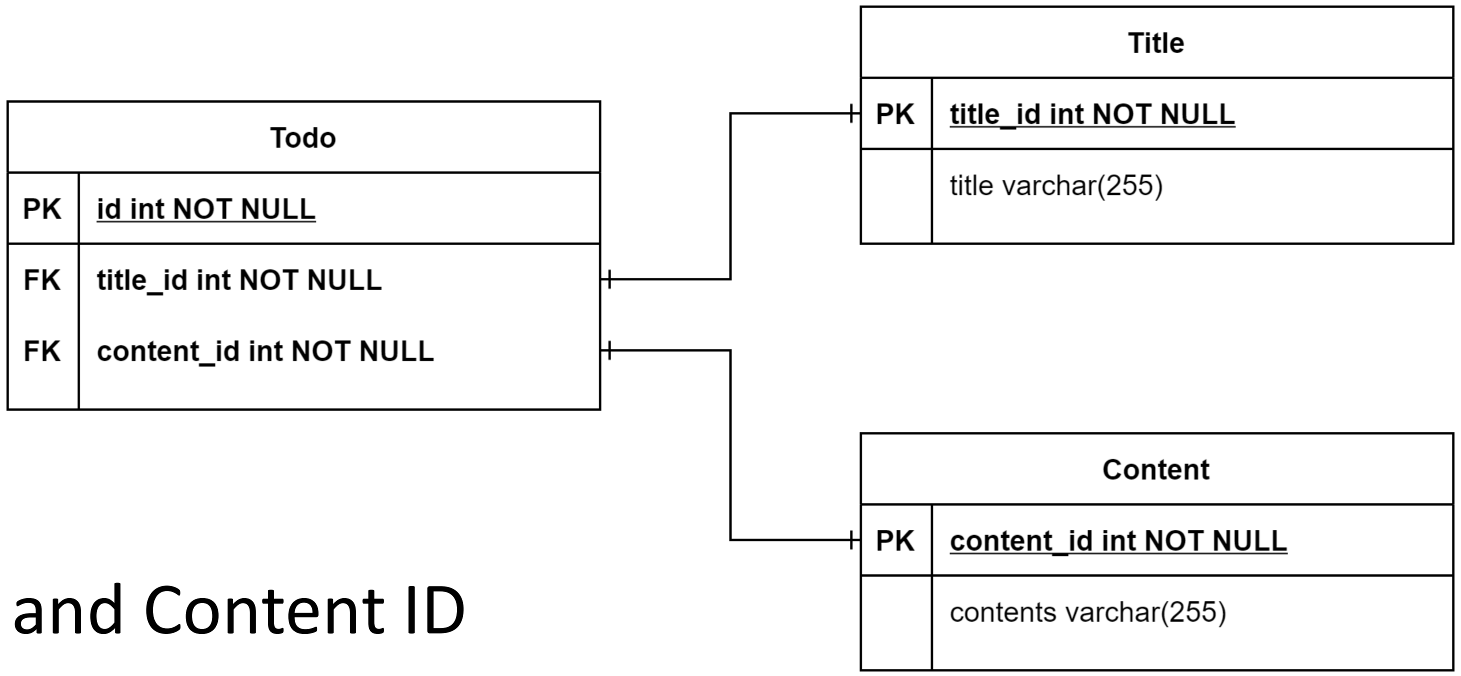
- Two tables
- The first, with ID and Contents ID
- The second with Contents ID, Title, and Contents
- One to One relationship between the two



Didn’t fix the problem


Object:Contents DTO
field1 = Title
field2 = Content
Content Mapper
Map to Content
Map to DTO
Object:Contents
field1 = Content ID
field2 = Title
field3 = Content
Todo Mapper
Map to Todo
Map to DTO
Object:Todo
field1 = ID
field2 = Content ID
Object:Todo DTO
field1 = ID
field2 = Title
field3 = Content
<<Interface>>
Contents Repository
+ method1(Todo): create
+ method2(Todo): find by id
+ method3(Todo): find all
+ method4(Todo): save
+ method5(Todo): delete by id
+ method6(Todo): delete all
Todo Controller
+ Service
Create
Read All
Read By ID
Update
Delete
Todo Service
+ Repository
+ Mapper
Create
Read All
Read By ID
Update
Delete
<<Interface>>
Todo Repository
+ method1(Todo): create
+ method2(Todo): find by id
+ method3(Todo): find all
+ method4(Todo): save
+ method5(Todo): delete by id
+ method6(Todo): delete all

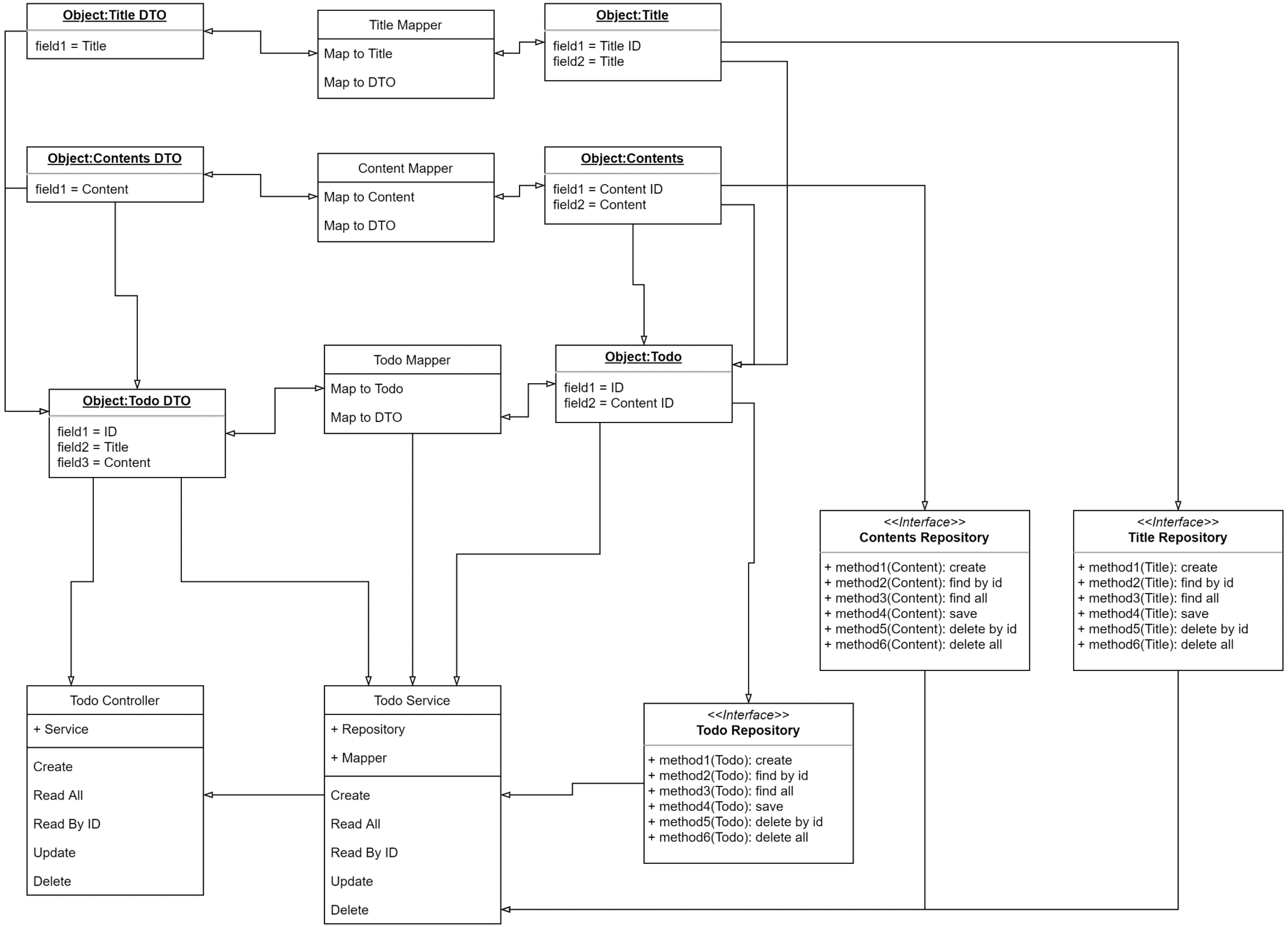
Plan



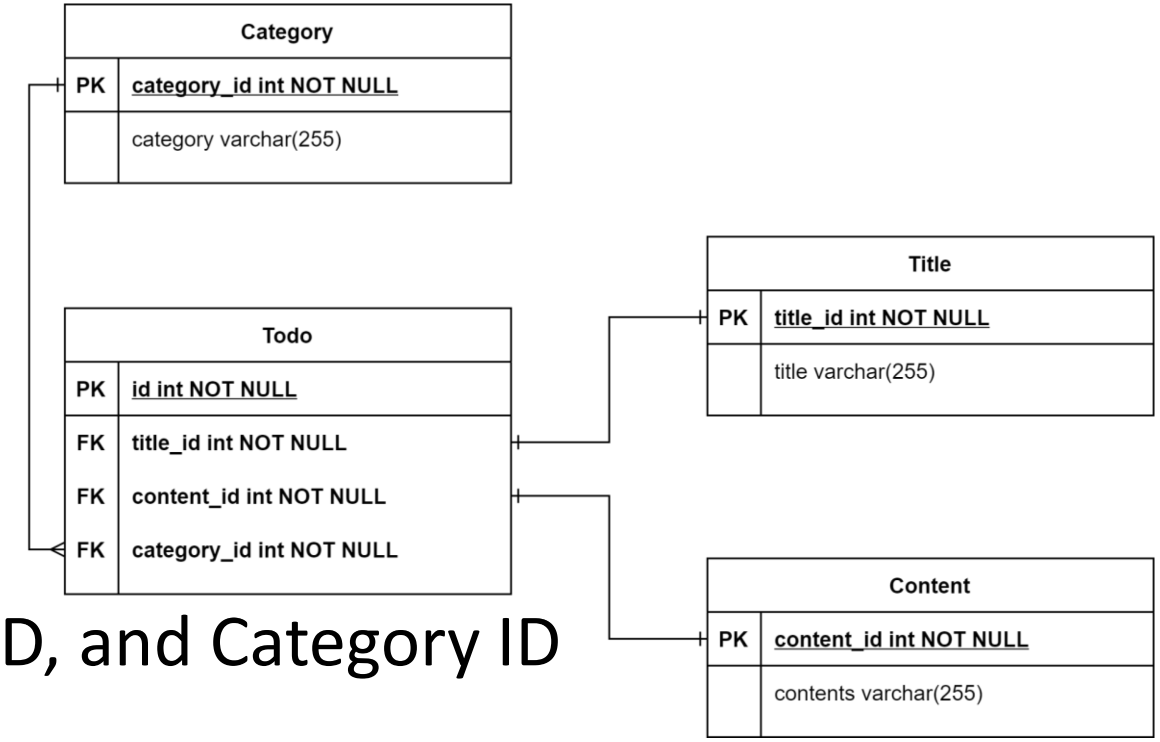
My third iteration:

- Three tables
- The first, with ID, Title ID, and Content ID
- The second with Title ID and Title
- The third with Contents ID and Contents
- One to One relationship between the first and the others

Null record problem now fixed

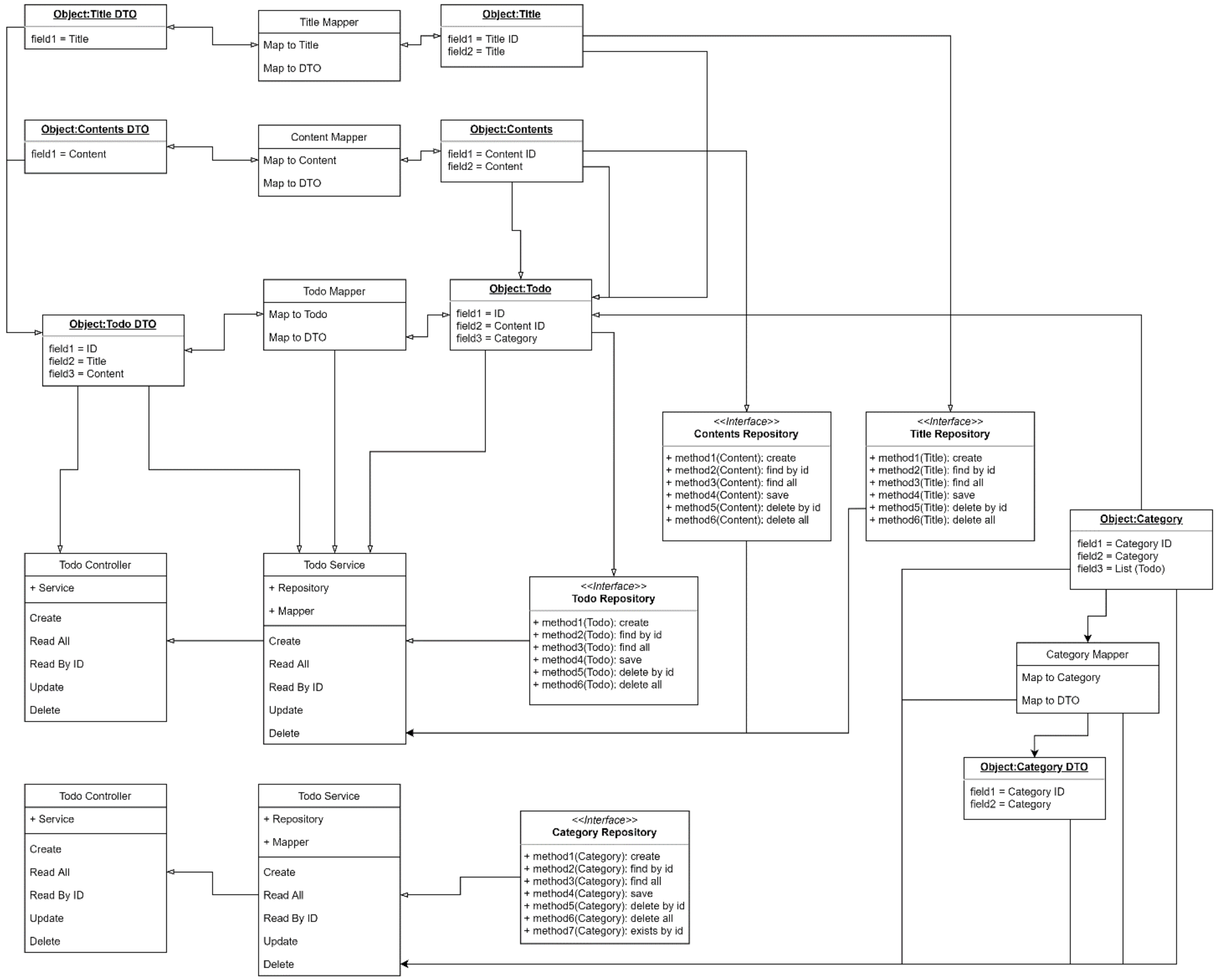

Object:Title DTO
field1 = Title
Title Mapper
Map to Title
Map to DTO
Object:Title
field1 = Title ID
field2 = Title
Object:Contents DTO
field1 = Content
Content Mapper
Map to Content
Map to DTO
Object:Contents
field1 = Content ID
field2 = Content
Todo Mapper
Map to Todo
Map to DTO
Object:Todo
field1 = ID
field2 = Content ID
Object:Todo DTO
field1 = ID
field2 = Title
field3 = Content
<<Interface>>
Contents Repository
+ method1(Content): create
+ method2(Content): find by id
+ method3(Content): find all
+ method4(Content): save
+ method5(Content): delete by id
+ method6(Content): delete all
<<Interface>>
Title Repository
+ method1(Title): create
+ method2(Title): find by id
+ method3(Title): find all
+ method4(Title): save
+ method5(Title): delete by id
+ method6(Title): delete all
Todo Controller
+ Service
Create
Read All
Read By ID
Update
Delete
Todo Service
+ Repository
+ Mapper
Create
Read All
Read By ID
Update
Delete
<<Interface>>
Todo Repository
+ method1(Todo): create
+ method2(Todo): find by id
+ method3(Todo): find all
+ method4(Todo): save
+ method5(Todo): delete by id
+ method6(Todo): delete all

Plan



My fourth iteration:

- Four tables
- The first with ID, Title ID, Content ID, and Category ID
- The second with Title ID and Title
- The third with Contents ID and Contents
- The fourth with Category ID and Category
- One to One relationship between Todo, and Title and Content
- One to Many relationship between Todo and Category


Object:Title DTO
field1 = Title
Title Mapper
Map to Title
Map to DTO
Object:Title
field1 = Title ID
field2 = Title
Object:Contents DTO
field1 = Content
Content Mapper
Map to Content
Map to DTO
Object:Contents
field1 = Content ID
field2 = Content
Todo Mapper
Map to Todo
Map to DTO
Object:Todo
field1 = ID
field2 = Content ID
field3 = Category
Object:Todo DTO
field1 = ID
field2 = Title
field3 = Content
<<Interface>>
Contents Repository
+ method1(Content): create
+ method2(Content): find by id
+ method3(Content): find all
+ method4(Content): save
+ method5(Content): delete by id
+ method6(Content): delete all
<<Interface>>
Title Repository
+ method1(Title): create
+ method2(Title): find by id
+ method3(Title): find all
+ method4(Title): save
+ method5(Title): delete by id
+ method6(Title): delete all
Object:Category
field1 = Category ID
field2 = Category
field3 = List (Todo)
Todo Controller
+ Service
Create
Read All
Read By ID
Update
Delete
Todo Service
+ Repository
+ Mapper
Create
Read All
Read By ID
Update
Delete
<<Interface>>
Todo Repository
+ method1(Todo): create
+ method2(Todo): find by id
+ method3(Todo): find all
+ method4(Todo): save
+ method5(Todo): delete by id
+ method6(Todo): delete all
Category Mapper
Map to Category
Map to DTO
Object:Category DTO
field1 = Category ID
field2 = Category
Todo Controller
+ Service
Create
Read All
Read By ID
Update
Delete
Todo Service
+ Repository
+ Mapper
Create
Read All
Read By ID
Update
Delete
<<Interface>>
Category Repository
+ method1(Category): create
+ method2(Category): find by id
+ method3(Category): find all
+ method4(Category): save
+ method5(Category): delete by id
+ method6(Category): delete all
+ method7(Category): exists by id

Database

- Todo database contains 4 tables: Todo, Title, Content, Category

- Todo: ID, Title ID, Content ID, Category ID
- Title: Title ID, Title

  One to One to Title ID in Todo
- Content: Content ID, Content

  One to One with Content ID in Todo
- Category: Category ID, Category

  One to Many with Category ID in Todo

Technologies used

- Visual Studio Code as IDE
- Java and Spring for the backend
- HTML, CSS, and JavaScript for the frontend
- Maven as build tool
- Git and GitHub for version tracking
- Jira for Kanban board and issue tracking
- JUnit 5 and Mockito for unit and integration testing
- Selenium for frontend testing
- JaCoCo and Extent Reports for test reports
- MySQL Server (local instance) and H2 to host database
- SonarQube and SonarLint for bug and code smell detection

Risk Assessment

		Impact				
		Negligible	Acceptable	Major	Problematic	Catastrophic
Likelihood	75% - 100%	A1	B1	C1	D1	E1
	50% - 74%	A2	B2	C2	D2	E2
	25% - 49%	A3	B3	C3	D3	E3
	0% - 24%	A4	B4	C4	D4	E4

This risk assessment shows some of the things I thought may be able to go wrong.

Each risk is detailed, with mitigation strategies and general advice to ignore such a situation, as well as a risk code.

Risks	Description	Response	Objective	Risk Code
Power cut	Power loss to computer means the project can't be worked on	Keep a backup on a laptop	Make sure a computer is always available to use	D4
Internet down	Internet down, no network communication	Use hotspot or use offline	Keep a system in place to continue working on the project even without internet	C4
Time underestimated on certain aspects	Some aspects of the project require more time than initially calculated	Research topics and incorporate them quickly	Do extensive research into all technologies and dependencies required for the project	B2
Hardware failure	Computer hardware failure, unable to use computer	Keep a backup on a working computer	Make sure all hardware on all computers is functional and maintained	E4
Database/Codebase deletion or mass modification	Project files or records from database may be accidently deleted	Keep backups of all code files and databases	Daily backups of all changes of the day will ensure no major changes are lost	D2
Strain Injuries	Working for too long may induce strain injuries	Keep moving throughout the day and take breaks	Make sure working position is confortable and regular breaks are taken	D3
Complicated source code	The blueprint code is too complicated and needs greater understanding to build upon	Take longer to look through the project and understand its function	Make sure every aspect of the project is understood before starting	B2
Features won't be completed in timeframe	Some features won't be completed in a week	Make a priority list of features, and implement them accordingly	Make sure enough time is allocated for the most important features	C2
Locked out of Jira	Jira canot be accessed anymore, and issues can't be tracked	Save an offline backup of issues, and authenticate multiple accoutns with the Jira site	Make sure isssues are available to use, otherwise project will have to be started from scratch	E4

Risks I faced

- On the first day, I lost access to my Jira issue tracking. Although I had not started much, there was a real risk of losing all the progress I had made
- Throughout the project, I had underestimated the time spent on certain features: Selenium testing, DB relationships working correctly, database updating
- Many of the features I wanted to include could not be completed in the timeframe. Therefore, features were prioritised according to the specification.

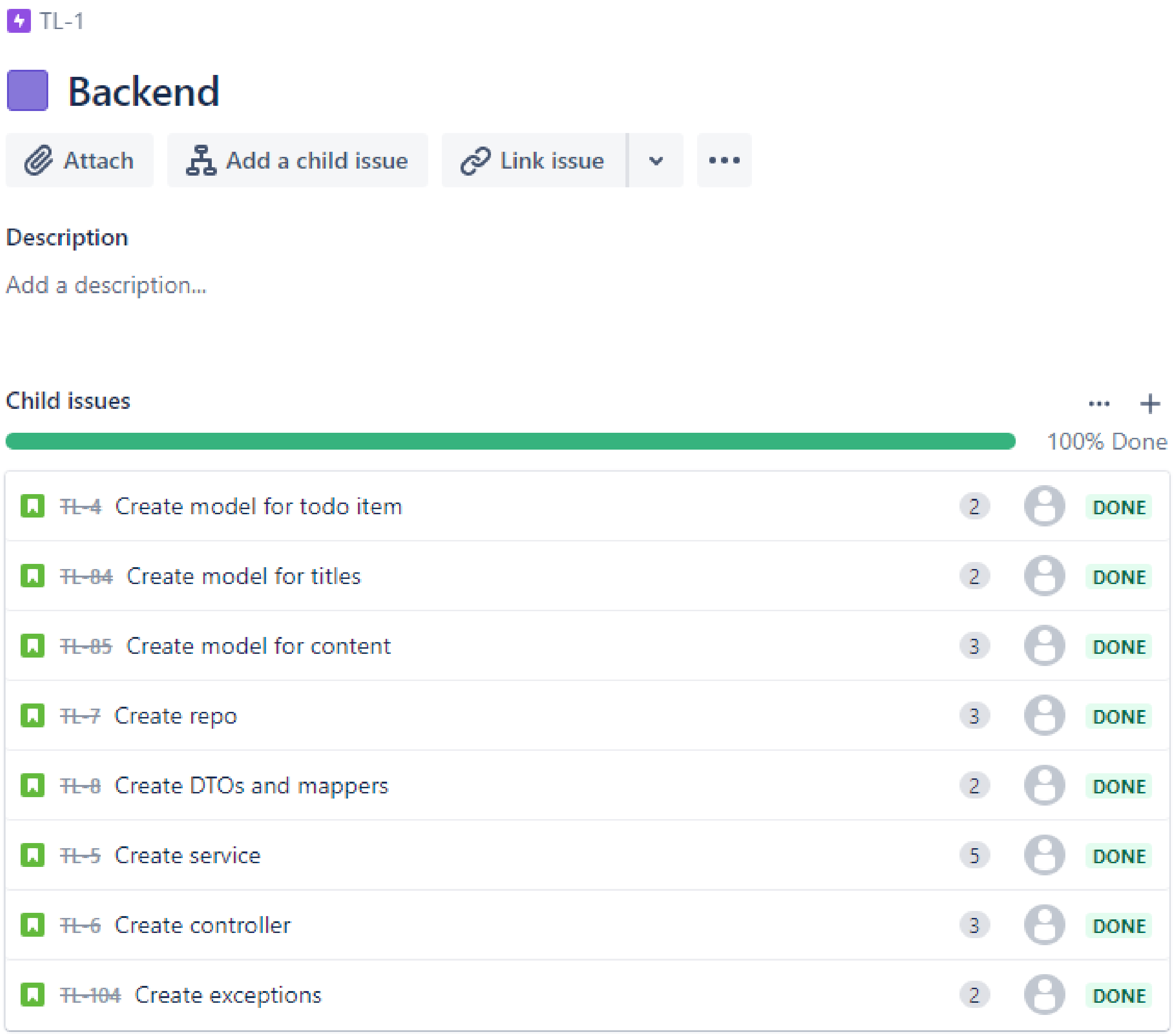
Issue tracking

I used Jira’s Next-Gen projects for issue tracking

- 4 main epics: Database Creation, App Implementation, Testing, User Management
- Each story was a collection of tasks for one broad function of the program:

  As a user, when I enter a new To Do Item, I want it to be stored in the database

- Smart commits and automation were used to link commits, branches, and pull requests to Tasks, User Stories, and Epics respectively.



TL-1

Backend

Attach | Add a child issue | Link issue

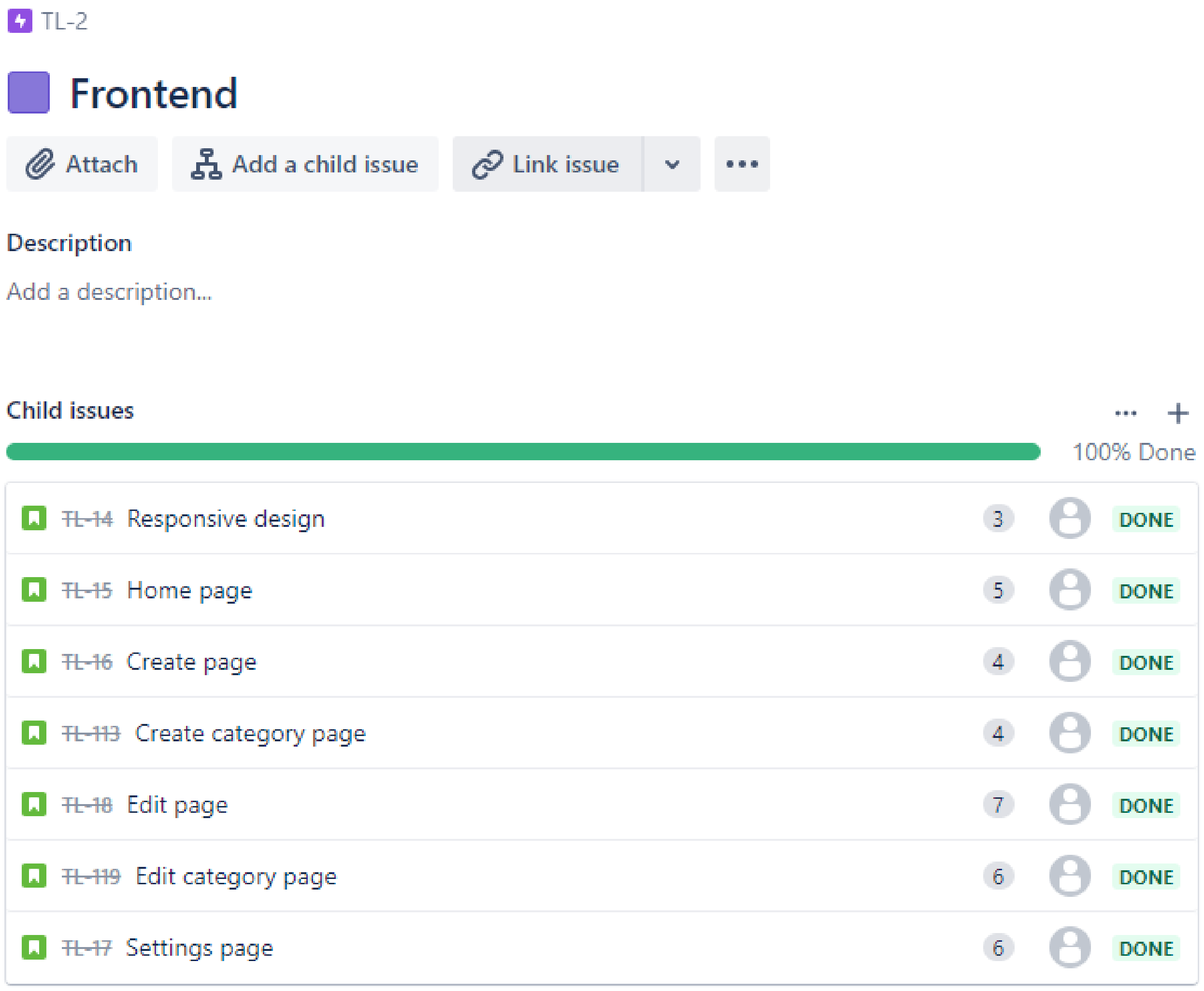
Description

Add a description...

Child issues

100% Done

~~TL-4~~	Create model for todo item	2	DONE
~~TL-84~~	Create model for titles	2	DONE
~~TL-85~~	Create model for content	3	DONE
~~TL-7~~	Create repo	3	DONE
~~TL-8~~	Create DTOs and mappers	2	DONE
~~TL-5~~	Create service	5	DONE
~~TL-6~~	Create controller	3	DONE
~~TL-104~~	Create exceptions	2	DONE



TL-2

Frontend

Attach | Add a child issue | Link issue

Description

Add a description...

Child issues

100% Done

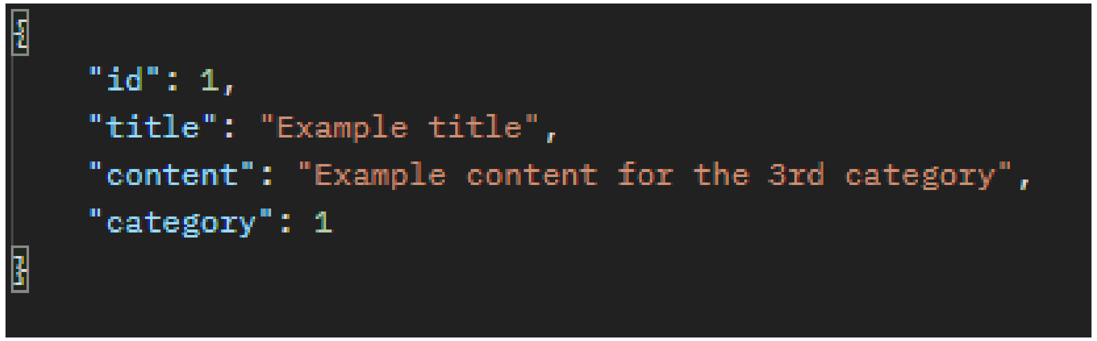
~~TL-14~~	Responsive design	3	DONE
~~TL-15~~	Home page	5	DONE
~~TL-16~~	Create page	4	DONE
~~TL-113~~	Create category page	4	DONE
~~TL-18~~	Edit page	7	DONE
~~TL-119~~	Edit category page	6	DONE
~~TL-17~~	Settings page	6	DONE

Sprint plans

- Throughout the time allocated, I had 3 sprints of varying lengths:
- Sprint 1 lasted 4 days, where the backend was created.
- Sprint 2 lasted 3 days, where the frontend was created
- Sprint 3 lasted 2 days, where tests were written for the frontend and backend

Sprint 1

- The backend has 4 model and DTO classes: Todo, Title, Content, and Category
- Title and Content are referenced by the Todo model, but never get used by the Todo service
- Todo DTO returns the text from Title and Content
- Category has its own Service and Controller classes
- If Unsorted doesn’t exist in the database at app start, it gets created.



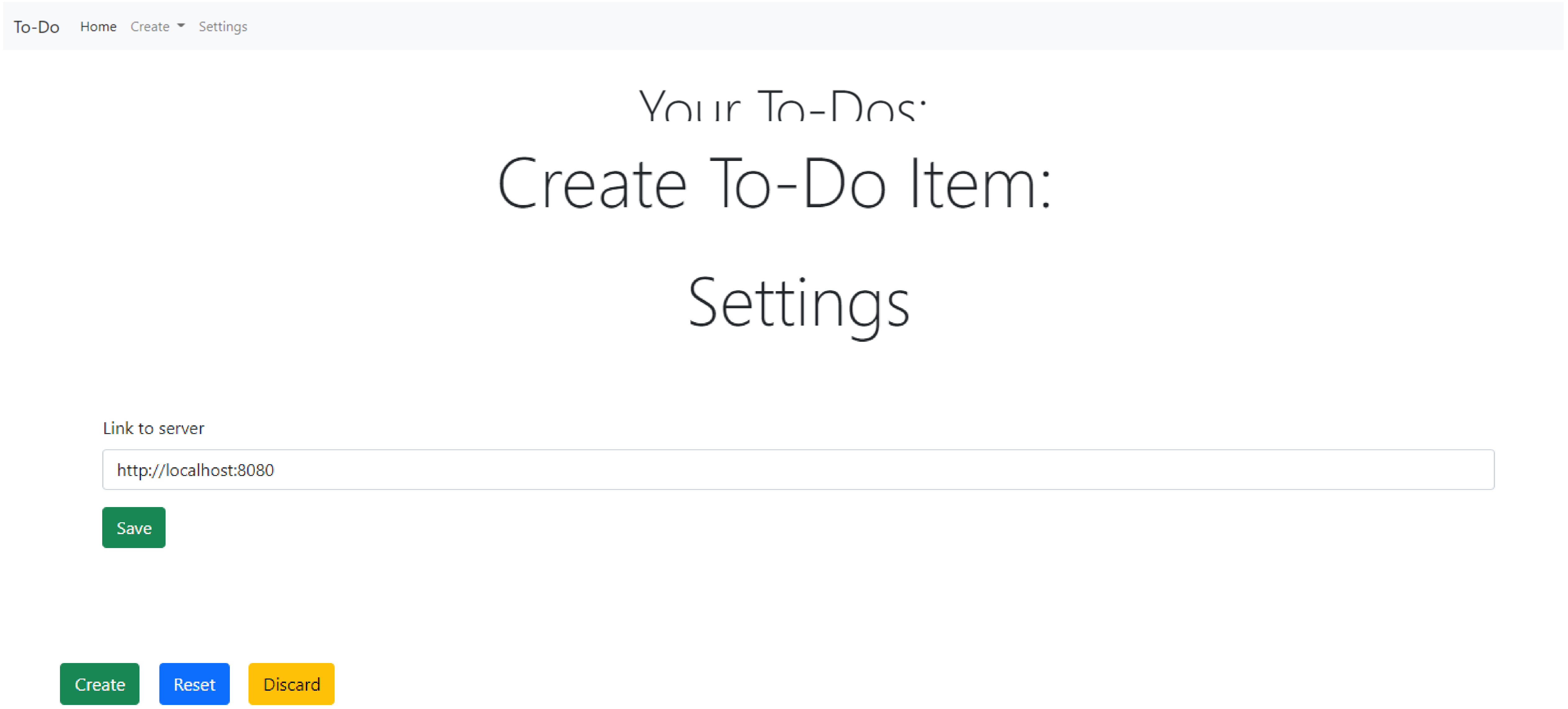
```
{
    "id": 1,
    "title": "Example title",
    "content": "Example content for the 3rd category",
    "category": 1
}
```

Review and Retro

- The backend did not take as long as anticipated
- Search by name and To Do items in multiple categories were left behind, would require even more tables
- Having foreign keys in Title and Content tables instead of Todo was the original plan
    - But Spring wasn’t creating the relationships, so they were moved to the Todo table

Sprint 2

- The frontend has a homepage, showing all categories and To Dos on one page. Each To Do is nested under a category.
- All communicating functions are async, so loaders are implemented
- The backend URL gets stored as a cookie, and is generated if there isn’t one on every start
- A Nav bar allows to navigate between Home, Create Category and To Do, and Settings
- Categories and To Dos on the home page are clickable to edit
- Settings allows you to set the backend URL, in case its hosted somewhere else.



Your To-Dos:

Create To-Do Item:

Settings

Link to server

http://localhost:8080

Review and Retro

- Most features planned out for the frontend were incorporated
- The frontend contains all basic CRUD functionality to communicate with the backend
- Also, async functions allowed for loaders, adding to the polished look
- Features I wanted to include are a Search function and user management
- Also, tidying up the UI of the frontend

Sprint 3

- 97.9% test coverage of backend achieved
- JUnit and Mockito tests of Models, Mappers, DTOs, Repositories, Services and Controllers
- Integration tests for Controller and Service, and Service and Repository
- SonarQube and SonarLint VS Code integration were used to refactor the code and reduce code smells

Sprint 3

- Selenium tests for all aspects of the frontend
- Tests for creating, updating, deleting, reading, setting the backend URL, and navigating between pages
- Extent report generated for each test, with screenshots of the webpages attached

Review and Retro

- Backend testing had quite a few problems, where one was finding a correct implementation of Cascade on Delete
- However, frontend testing with Selenium came with a number of problems:
  - Driver couldn’t be closed after each test
  - After every test, Spring wanted to restart the database and the Tomcat server.
  - Cookies can’t be set when browsing local HTML files, so frontend has to be hosted
  - Whilst creating property to set backend link, Spring Boot refused to read it

- For the future, I would find a way to refactor the tests into separate files.

Demo

Any questions?

Repo located at:

https://github.com/alihamzamohammed/IMS-Starter/