Todo-App

Jared Moore — December, 2020

Technologies Uses - Server Side

- **NodeJS** (Server)
- **Express** (Web Server)
- **TypeScript** (Language)
- **TypeORM** (Database Connector)
- **GraphQL** (API Data Layer)
- **Apollo Server** (Graphql Server)

Technologies Uses - Client Side

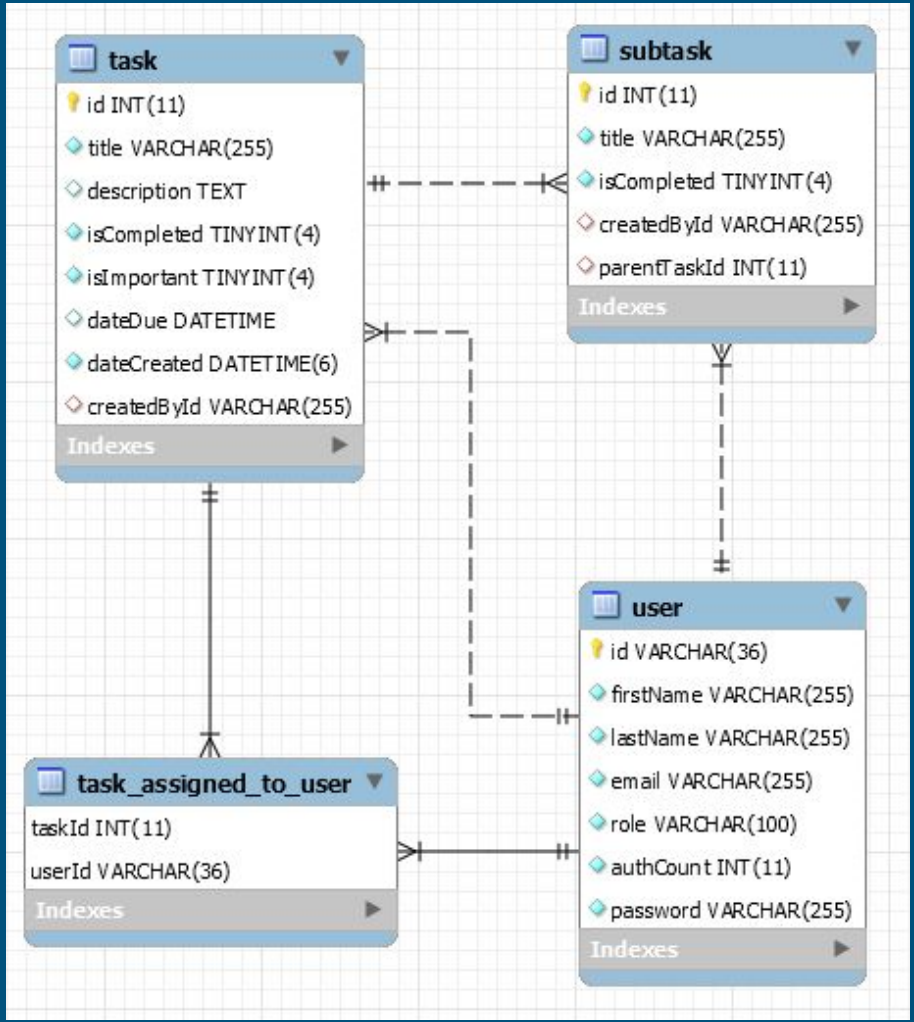
- **React** (Web Framework)
- **TypeScript** (Language)
- **AntD** (React UI Library)
- **Apollo Client** (GraphQL Connector)

Features

- Create Task
  - Set due date
  - Set description
  - Set importance
- Create Subtasks
  - Break down task into smaller subtasks
- Assign users to a task you created, share tasks
- Filter tasks
  - By due date
  - By important
  - By personal tasks
  - By assigned tasks

Database Diagram

- Many tasks to many users
- One user to one subtask (createdById)
- One subtask to task, tasks can have zero or more subtasks



Docker Environment

- Uses docker to create a MySQL database that is consistent across deployments
- Simplifies the development and production environment because the database is constant across all platforms and run times

```yaml
version: '3.5'

services:
  mysql:
    container_name: mysql
    image: mysql:5.7.27
    command: --default-authentication-plugin=mysql_native_password
    environment:
      MYSQL_USER: user
      MYSQL_PASSWORD: password
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: db
    ports:
      - 3306:3306
    expose:
      - 3306
    volumes:
      - mysql:/var/lib/mysql
    networks:
      - mysql
    restart: always
```

User Table, Mapped to Entity

- The fields in the user class match the columns from the user table in MySQL
- The properties are marked with decorators giving more control over the data within the program

```
@Entity() @ObjectType()
export class User extends BaseEntity {
  @Field(() => ID)
  @PrimaryGeneratedColumn('uuid')
  id: string

  @Field()
  @Column()
  firstName: string

  @Field()
  @Column()
  lastName: string

  @Field()
  name(@Root() parent: User): string {
    return `${parent.firstName} ${parent.lastName}`
  }

  @Field()
  @Column({ unique: true })
  email: string

  @Field(() => UserRole)
  @Column('varchar', { default: UserRole['USER'], length: 100 })
  role: UserRole

  // JWT Auth
  @Field({ nullable: true })
  @Column({ default: 0 })
  authCount: number

  // User Auth
  @Column()
  password: string
}
```

Create User

- Get user fields from web client
- Hash user password
- Store user information and hashed password on database

```
import { Resolver, Mutation, Arg } from "type-graphql";
import { User } from "../../entity/User";
import bcrypt from "bcryptjs";

import { RegisterInput } from "./register/RegisterInput";
import { sendConfirmationEmail } from "../../helpers/sendEmail";

@Resolver()
export class RegisterResolver {
  @Mutation(() => User)
  async register(@Arg('data') {
    firstName,
    lastName,
    email,
    password
  }: RegisterInput): Promise<User> {
    const hashedPassword = await bcrypt.hash(password, 12);

    const user = await User.create({
      firstName,
      lastName,
      email,
      password: hashedPassword
    }).save();

    sendConfirmationEmail(user);

    return user;
  }
}
```

Login User

- User submits email and password
- Find user by email
- Hash the password submitted during login and compare to hashed value it the database
- Set JWT token in the user's browser to persist login credentials

```
1   import { Resolver, Mutation, Arg, Ctx } from "type-graphql";
2   import { User } from "../../entity/User";
3   import bcrypt from "bcryptjs";
4   import { MyContext } from "../../ts/context";
5   import { createTokens } from "../../helpers/auth";
6
7   @Resolver()
8   export class LoginResolver {
9     @Mutation(() => User, { nullable: true })
10    async login(
11      @Arg('email') email: string,
12      @Arg('password') password: string,
13      @Ctx() ctx: MyContext
14    ): Promise<User | null> {
15      const user = await User.findOne({ where: { email } });
16      if (!user) throw "email and or password are invalid";
17
18      const valid = await bcrypt.compare(password, user.password);
19      if (!valid) throw "email and or password is invalid";
20
21      const tokens = createTokens(user);
22
23      ctx.res.cookie("refresh-token", tokens.refreshToken);
24      ctx.res.cookie("access-token", tokens.accessToken);
25
26      return user;
27    }
28  }
```

Create Task Mutation

- TypeORM can provide a layer of abstraction on top of the database, which allows for quicker programming

```
@Authorized([UserRole['USER']])
@Mutation(() => Task)
async createTask(@Arg('data') data: TaskInput, @Ctx() ctx: MyContext): Promise<Task> {
  const user = await User.findOne(ctx.req.userId)

  if (!user) throw new Error('Please login')

  const task = await Task.create({ ...data }).save().catch(err => { throw err })
  task.createdBy = user
  task.subtasks = []
  task.assignedTo = []
  await task.save()

  return task
}
```

Tasks Query, Raw SQL

- Select Tasks
- Join subtasks, createdBy, and assignedTo
- Filter records based on passed parameters
  - `dateAscending?: boolean`
  - `isImportant?: boolean`
  - `isCompleted?: boolean`
  - `title?: string`
  - `taskOwner?: TaskFiltersOwner`
  - `dateDue?: Date`
  - `dateDueOperator?: TaskFiltersDateOperator`

```
@Authorized([UserRole['USER']])
@Query(() => [Task])
async tasks(@Ctx() ctx: MyContext, @Arg('filters', { nullable: true }) filters: TaskFilters): Promise<Task[]> {
  const user = await User.findOne({ where: { id: ctx.req.userId } }).catch(err => { throw err })
  if (!user) throw new Error("User not found by id")

  // query builder
  const qb = getConnection().createQueryBuilder()
  qb.select("task")
  qb.from(Task, "task")

  // relations
  qb.leftJoinAndSelect('task.subtasks', "subtasks", "subtasks.parentTaskId = task.id")
  qb.leftJoinAndSelect('task.createdBy', "createdBy", "createdBy.id = task.createdById")
  qb.leftJoinAndSelect('task.assignedTo', "assignedTo", "assignedTo.id IN (SELECT u.userId FROM task_assigned_to_user u WHERE u.taskId = task.id)")

  // owner filter
  if (filters.taskOwner === TaskFiltersOwner['USER_ASSIGNED'])
    qb.where(`('${user.id}' IN (SELECT u.userId FROM task_assigned_to_user u WHERE u.taskId = task.id) OR task.id IN (SELECT u.taskId FROM task_assigned_to_user u WHERE u.taskId = task.id))`)
  else if (filters.taskOwner === TaskFiltersOwner['USER_CREATED']) {
    qb.where(`(task.createdById = '${user.id}') AND (task.id NOT IN (SELECT u.taskId FROM task_assigned_to_user u WHERE u.taskId = task.id))`)
  } else {
    qb.where(`(task.createdById = '${user.id}' OR '${user.id}' IN (SELECT u.userId FROM task_assigned_to_user u WHERE u.taskId = task.id))`)
  }
  // TODO: filter by task owner

  // is important
  if (typeof filters.isImportant === "boolean")
    qb.andWhere(`task.isImportant = ${filters.isImportant ? 1 : 0}`)

  // is completed
  if (typeof filters.isCompleted === "boolean")
    qb.andWhere(`task.isCompleted = ${filters.isCompleted ? 1 : 0}`)

  // title
  if (typeof filters.title === "string")
    qb.andWhere(`task.title LIKE '%${filters.title.trim().toLowerCase() ? 1 : 0}%'`)

  // date due
  if (filters.dateDue && filters.dateDueOperator)
    qb.andWhere(`DATE_FORMAT(task.dateDue, "%Y-%m-%d") ${filters.dateDueOperator} "${javascriptDateToSQLDateYMD(filters.dateDue)}"`)

  // order
  if (!filters.dateAscending) qb.orderBy('task.dateDue', 'DESC')
  else qb.orderBy('task.dateDue', 'ASC')

  const tasks = await qb.getMany().catch(err => { throw err })

  return tasks
}
```

Task Table Data

Limit to 1000 rows

```
1 •  SELECT * FROM db.task;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	title	description	isCompleted	isImportant	dateDue	dateCreated	createdById
14	Personal Task		0	0	2020-11-30 13:32:50	2020-11-30 04:55:40.185168	09e4cfd9-bca1-4898-ba67-e336d8f48ff1
15	Late Task		0	0	2020-10-01 00:22:58	2020-11-30 05:22:49.071812	8411882f-7920-4112-996d-f5114814e248
16	Today Task		0	1	2020-11-30 00:34:51	2020-11-30 05:22:51.756774	8411882f-7920-4112-996d-f5114814e248
17	Future Task		0	0	2020-12-01 00:23:09	2020-11-30 05:22:57.604838	8411882f-7920-4112-996d-f5114814e248
18	Shared		0	0	NULL	2020-11-30 06:12:10.722250	09e4cfd9-bca1-4898-ba67-e336d8f48ff1
19	Go Christmas Shopping		0	0	NULL	2020-12-01 18:33:25.594918	09e4cfd9-bca1-4898-ba67-e336d8f48ff1
20	Do Quiz		0	1	2020-12-01 22:24:13	2020-12-02 03:24:02.537758	09e4cfd9-bca1-4898-ba67-e336d8f48ff1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

GraphQL Query

- GraphQL allows you to represent your data in simple queries
- It is a layer between the database and the client
- Response data always matches the query structure and it returned as JSON

PRETTIFY HISTORY http://localhost:3001/graphql

```
1 {
2   tasks(filters: {}) {
3     id
4     title
5     description
6     isCompleted
7     isImportant
8     dateDue
9   }
10 }
```


```
{
  "data": {
    "tasks": [
      {
        "id": 20,
        "title": "Do Quiz",
        "description": "",
        "isCompleted": false,
        "isImportant": true,
        "dateDue": "2020-12-02T03:24:13.000Z"
      },
      {
        "id": 17,
        "title": "Future Task",
        "description": "",
        "isCompleted": false,
        "isImportant": false,
        "dateDue": "2020-12-01T05:23:09.000Z"
      },
      {
        "id": 16,
        "title": "Today Task",
        "description": "",
        "isCompleted": false,
        "isImportant": true,
        "dateDue": "2020-11-30T05:34:51.000Z"
```

Example React Component

- Each HTML section of a web app is broken down to individual pieces
- Data is passed to the component and automatically updates

```
1  import { StarFilled, StarOutlined } from '@ant-design/icons';
2  import { Button } from 'antd';
3  import React from "react";
4
5  interface StarCheckboxProps {
6    isChecked: boolean
7    onClick: (args: { event: React.MouseEvent<HTMLElement, MouseEvent>, isChecked: boolean }) => void
8    style?: React.CSSProperties
9  }
10
11 export const StarCheckbox: React.FC<StarCheckboxProps> = props => <Button
12   className="icon-btn-lg"
13   shape="circle"
14   icon={props.isChecked ? <StarFilled /> : <StarOutlined />}
15   onClick={(e) => { if (props.onClick) props.onClick({ event: e, isChecked: !props.isChecked }) }}
16   style={{ color: props.isChecked ? "rgb(255 188 0 / 100%)" : "rgb(0 0 0 / 20%)", transition: 'ease 0.5s color' }}
17 />
```