Backend 1.5_CW Exercises

Exploring Reference and Populate:

Part 1: Understanding Reference and Populate

Let's begin by understanding reference and populate using a simple example involving Car and Maker Models.

Exercise: Using Reference between Car and Maker Models

In this exercise, we'll establish a reference relationship between the Car Model and the Maker Model.

1. Define the Maker Model with keys name(String), logo(String) and tagline(String).

    - Solution

```
const mongoose = require('mongoose')

const makerSchema = new mongoose.Schema({
  name: String,
  logo: String,
  tagline: String,
})

const Maker = mongoose.model('Maker', makerSchema)

module.exports = Maker
```

COPY

2. Update the previously made Car Model to include a reference to the Maker Model:

    - Solution

```
const mongoose = require('mongoose')

const carSchema = new mongoose.Schema({
  model: String,
  year: Number,
  maker: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Maker',
  },
})

const Car = mongoose.model('Car', carSchema)
```

```
module.exports = Car
```

COPY

- Explanation details

```
// Mentos Zindagi

// Define Maker once
{ _id: "3489y439834001", name: "Maruti Suzuki", logo: "somethig.com/pic.png", taglir

// And then refer to it
{
name: '800',
year: 2022,
maker: "3489y439834001"
}

{
        name: 'Omni',
year: 2022,
maker: "3489y439834001"
}

// Aam Zindagi

// Repeat Maker's object every time
{
_id: "3489y439834001",
        name: '800',
year: 2022,
maker: { name: "Maruti Suzuki", logo: "somethig.com/pic.png", tagline: "desh ki sava
}

{
        name: 'Omni',
year: 2022,
maker: { name: "Maruti Suzuki", logo: "somethig.com/pic.png", tagline: "desh ki sava
}

{
        name: 'Alto',
year: 2022,
maker: { name: "Maruti Suzuki", logo: "somethig.com/pic.png", tagline: "desh ki sava
}

{
        name: 'WagonR',
year: 2022,
maker: { name: "Maruti Suzuki", logo: "somethig.com/pic.png", tagline: "desh ki sava
}
```

COPY

3. Add data to Maker model and the Car model with reference to Maker model:

- Solution

```
async function addMaker(makerData) {
  try {
    // 1. Creating a new Maker
```

```
    const maker = new Maker(makerData)
    const newMaker = await maker.save()
    console.log('New maker created:', newMaker)

    // 2. Defining a new Car
    const carData = {
      model: 'Car Model XL',
      year: 2022,
      maker: newMaker._id, // 3. Using created maker's _id earlier
    }

    // 4. Create a new Car
    const car = new Car(carData)
    const newCar = await car.save()

    console.log('New Car:', newCar)
  } catch (error) {
    throw error
  }
}

const makerData = {
  model: 'Toyota',
  logo: 'maker_logo_url1',
  tagline: 'Quality Cars',
}
addMaker(makerData)
```

COPY

4. Create a function to retrieve a car and populate its maker details:

   - Solution https://replit.com/@tanaypratap/BE15CW-ex01

```
async function getCarWithMakerDetails(carId) {
  try {
    const carWithMaker = await Car.findById(carId).populate('maker')
    console.log('Car with maker details:', carWithMaker)
  } catch (error) {
    throw error
  }
}

// Example usage
getCarWithMakerDetails('your-car-id-here')
```

COPY

Part 2: Enhancing the Movie Model with Ratings and Reviews

Let's now enhance the Movie Model by adding user ratings and reviews.

Exercise: Updating Movie Model with User Ratings and Reviews

1. Update the Movie Model to include `reviews` field:

- Solution

```
const movieSchema = new mongoose.Schema({
  reviews: [
    {
      user: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
      },
      text: String, // Add a field for review text
    },
  ],
})

const Movie = mongoose.model('Movie', movieSchema)
```

COPY

1. Create a function to add a review to a movie:

https://replit.com/@tanaypratap/BE15CW-ex021

- Breakdown steps

1. Get the movie to be updated // findOneById
2. const movieToBeUpdated = await findOneById
3. movieToBeUpdated.reviews
4. `{ user: 'isissksk3787393', text: 'OMG! Loved it!' }`
5. `movieToBeUpdated.reviews.push({ user: 'isissksk3787393', text: 'OMG! Loved it!' })`
6. movieToBeUpdated.save()

COPY

- Solution

```
async function addRatingAndReview(movieId, userId, reviewText) {
  try {
    const movie = await Movie.findById(movieId)
    if (movie) {
      // Create a new review object with user and review text
      const review = {
        user: userId,
        text: reviewText,
      }
      movie.reviews.push(review)

      await movie.save()

      const updatedMovieWithReview = await Movie.findById(movieId).populate(
        'reviews.user',
        'username profilePictureUrl',
      )
      console.log('Updated movie with review:', updatedMovieWithReview)
    } else {
      throw new Error('Movie not found')
    }
  } catch (error) {
    throw error
```

```
  }
}

// Example usage
addRatingAndReview(
  'your-movie-id-here',
  'your-user-id-here',
  'A fantastic movie!',
)
```

COPY

Exercise: Querying Top and Bottom Ratings of a Movie

1. Create a function to retrieve the top 5 ratings and reviews of a movie:

https://replit.com/@tanaypratap/BE15CW-ex022

- Solution

```
async function getTopRatingsAndReviews(movieId) {
  try {
    const movie = await Movie.findById(movieId).populate('reviews')
    movie.ratings.sort((a, b) => b - a)
    return movie.ratings.slice(0, 5).map((rating, index) => ({
      rating,
      review: movie.reviews[index],
    }))
  } catch (error) {
    throw error
  }
}

// Example usage
async function getFiveTopRatingsAndReviews(movieId) {
  try {
    const topRatingsAndReviews = await getTopRatingsAndReviews(movieId)
    console.log('Top ratings and reviews:', topRatingsAndReviews)
  } catch (error) {
    console.log(error)
  }
}

getFiveTopRatingsAndReviews('your-movie-id-here')
```

COPY

1. Create a function to retrieve the bottom 5 ratings and reviews of a movie:

https://replit.com/@tanaypratap/BE15CW-ex023

- Solution

```
async function getBottomRatingsAndReviews(movieId) {
  try {
    const movie = await Movie.findById(movieId).populate('reviews')
    movie.ratings.sort((a, b) => a - b)
    return movie.ratings.slice(0, 5).map((rating, index) => ({
```

```
      rating,
      review: movie.reviews[index],
    }))
  } catch (error) {
    throw error
  }
}

// Example usage
async function getFiveBottomRatingsAndReviews(movieId) {
  try {
    const bottomRatingsAndReviews = await getBottomRatingsAndReviews(movieId)
    console.log('Bottom ratings and reviews:', bottomRatingsAndReviews)
  } catch (error) {
    console.log(error)
  }
}

getFiveBottomRatingsAndReviews('your-movie-id-here')
```

COPY

Exercise: Querying Reviews of a Movie

Create a function to retrieve the first 3 reviews of a movie, populated with user details:

https://replit.com/@tanaypratap/BE15CW-ex024

- Solution

```
async function getMovieReviewsWithUserDetails(movieId) {
  try {
    const movie = await Movie.findById(movieId).populate({
      path: 'reviews',
      populate: {
        path: 'user',
        select: 'username profilePictureUrl',
      },
    })
    const reviewsWithUserDetails = movie.reviews
      .slice(0, 3)
      .map((review) => ({
        reviewText: review.text,
        user: review.user,
      }))
    return reviewsWithUserDetails
  } catch (error) {
    throw error
  }
}

// Example usage
async function getThreeMovieReviewsWithUserDetails(movieId) {
  try {
    const reviewsWithUserDetails = await getMovieReviewsWithUserDetails(
      movieId,
    )
    console.log('Movie reviews with user details:', reviewsWithUserDetails)
```

```
  } catch (error) {
    console.log(error)
  }
}

getThreeMovieReviewsWithUserDetails('your-movie-id-here')
```