API Code

# drivers

## route.ts

import

{

NextResponse

}

from

'next/server';

import

mongoose,

{

model,

models

}

from

'mongoose';

import

{

connectToDatabase,

driverSchema

}

from

'@/utils/schema';

import

{

IDriver

}

from

'@/utils/interface';

import

{

verifyToken

}

from

'@/utils/auth';

const

Driver

=

models.Driver

||

model('Driver',

driverSchema);

export

async

function

GET(req

:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

drivers

=

await

Driver.find({user\_id

:

user}).lean().exec();

return

NextResponse.json({

drivers

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

export

async

function

POST(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

data

=

await

req.json();

//

Phone

number

validation

(10

digits

starting

with

7,

8,

or

9)

const

phoneRegex

=

/^[6789]\d{9}$/;

if

(data.contactNumber

!=

''

&&

!phoneRegex.test(data.contactNumber))

{

return

NextResponse.json({

message:

'Invalid

phone

number'

},

{

status:

400

});

}

const

newDriver:

IDriver

=

new

Driver({

user\_id

:

user,

driver\_id

:

data.driver\_id,

name:

data.name,

contactNumber

:

data.contactNumber,

balance

:

data.balance,

status

:

data.status

});

const

savedDriver

=

await

newDriver.save();

return

NextResponse.json({

message:

'Saved

Successfully',

data:

savedDriver

},

{

status:

200

});

}

catch

(error:

any)

{

console.error('Error

saving

Driver:',

error);

if

(error.name

===

'ValidationError')

{

return

NextResponse.json({

message:

'Validation

Error',

details:

error.message

},

{

status:

400

});

}

else

if

(error.name

===

'MongoError'

&&

error.code

===

11000)

{

return

NextResponse.json({

message:

'Duplicate

Key

Error',

details:

error.message

},

{

status:

409

});

}

else

{

return

NextResponse.json({

message:

'Internal

Server

Error',

details:

error.message

},

{

status:

500

});

}

}

}

# drivers/create

## route.ts

import

{

NextResponse

}

from

'next/server';

import

mongoose,

{

model,

models

}

from

'mongoose';

import

{

connectToDatabase,

driverSchema

}

from

'@/utils/schema';

import

{

IDriver

}

from

'@/utils/interface';

import

{

verifyToken

}

from

'@/utils/auth';

const

Driver

=

models.Driver

||

model('Driver',

driverSchema);

export

async

function

GET(req

:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

drivers

=

await

Driver.find({user\_id

:

user}).select(['name',

'driver\_id','status']).lean().exec();

return

NextResponse.json({

drivers

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# drivers/[driverId]

## route.ts

import

{

connectToDatabase,

driverSchema

}

from

"@/utils/schema";

import

{model,

models

}

from

"mongoose";

import

{v4

as

uuidv4}

from

'uuid'

const

Driver

=

models.Driver

||

model('Driver',

driverSchema)

import

{

NextResponse

}

from

'next/server';

import

{

IDriver

}

from

'@/utils/interface';

import

{

verifyToken

}

from

"@/utils/auth";

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

driverId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

driverId

}

=

params;

try

{

await

connectToDatabase();

const

driver

=

await

Driver.findOne({user\_id

:

user,

driver\_id

:

driverId}).exec();

if

(!driver)

{

return

NextResponse.json({

message:

'Driver

not

found'

},

{

status:

404

});

}

return

NextResponse.json(driver,

{

status:

200

});

}

catch

(err:

any)

{

console.log(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

export

async

function

PUT(req:

Request,

{params}:

{params

:

{driverId:

string}}){

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{driverId}

=

params

const

data

=

await

req.json()

try{

await

connectToDatabase();

const

driver:

IDriver

=

await

Driver.findOne({user\_id

:

user

,

driver\_id

:

driverId}).exec();

driver.balance

=

driver.balance

+

data.got

-

data.gave

driver.accounts.push({

account\_id

:

'account'

+

uuidv4(),

date

:

data.date,

reason

:

data.reason,

gave

:

data.gave,

got

:

data.got

})

driver.save()

return

NextResponse.json({driver

:

driver},

{status

:

200})

}catch(err){

console.log(err)

}

}

export

async

function

DELETE(req:

Request,

{

params

}:

{

params:

{

driverId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

driverId

}

=

params;

try

{

await

connectToDatabase();

const

foundDriver

=

await

Driver.findOne({

user\_id:

user,

driver\_id:

driverId

});

if

(foundDriver.status

==

'On

Trip')

{

return

NextResponse.json({

message:

'Driver

On

Trip

Cannot

Delete'

},

{

status:

400

});

}

const

driver

=

await

Driver.findOneAndDelete({

driver\_id:

driverId

});

if

(!driver)

{

return

NextResponse.json({

message:

'Driver

not

found'

},

{

status:

404

});

}

return

NextResponse.json({

message:

'Driver

Deleted'

},

{

status:

200

});

}

catch

(err:

any)

{

return

NextResponse.json({

message:

err.message

},

{

status:

500

});

}

}

export

async

function

PATCH(req:

Request,

{

params

}:

{

params:

{

driverId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

const

{

driverId

}

=

params;

const

{

name,

contactNumber,

status

}

=

await

req.json();

console.log(`${status}`)

await

connectToDatabase();

//

Ensure

this

function

is

properly

defined

and

imported

const

driver

=

await

Driver.findOne({

user\_id

:

user,

driver\_id:

driverId

});

if

(!driver)

{

return

NextResponse.json({

message:

'No

Driver

Found'

},

{

status:

404

});

}

if

(name)

driver.name

=

name;

if

(contactNumber)

driver.contactNumber

=

contactNumber;

if

(status)

driver.status

=

status;

await

driver.save();

return

NextResponse.json({

driver:

driver

},

{

status:

200

});

}

catch

(err:

any)

{

console.log(err)

return

NextResponse.json({

message:

err.message

},

{

status:

500

});

}

}

# drivers/[driverId]/accounts/[accountId]

## route.ts

import

{

connectToDatabase,

driverSchema

}

from

"@/utils/schema";

import

{model,

models

}

from

"mongoose";

const

Driver

=

models.Driver

||

model('Driver',

driverSchema)

import

{

NextResponse

}

from

'next/server';

import

{

IDriver

}

from

'@/utils/interface';

import

{

verifyToken

}

from

"@/utils/auth";

export

async

function

DELETE(req:

Request,{

params

}:

{

params:

{

driverId:

string;

accountId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

//

const

{

driverId,

accountId

}

=

params;

const

url

=

req.url

const

url\_arr

=

url?.split('/')

const

accountId

=

url\_arr[7]

const

driverId

=

url\_arr[5]

try

{

await

connectToDatabase();

//

Find

the

driver

document

by

driverId

const

driver

=

await

Driver.findOne({

user\_id:

user,

driver\_id:

driverId

});

if

(!driver)

{

return

NextResponse.json({

message:

'Driver

not

found'

},

{

status:

404

});

}

//

Filter

out

the

account

with

the

specified

accountId

from

the

accounts

array

driver.accounts

=

driver.accounts.filter((account

:

any)

=>

account.account\_id

!==

accountId);

//

Save

the

updated

driver

document

await

driver.save();

return

NextResponse.json({

driver

},

{

status:

200

});

}

catch

(error)

{

console.error('Failed

to

delete

account

detail:',

error);

return

NextResponse.json({

error:

'Failed

to

delete

account

detail'

},

{

status:

500

});

}

}

export

async

function

PATCH(req:

Request,

{

params

}:

{

params:

{

driverId:

string;

accountId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

driverId,

accountId

}

=

params;

const

data

=

await

req.json();

console.log(driverId)

console.log(accountId)

try

{

await

connectToDatabase();

//

Find

the

driver

document

by

driverId

and

user\_id

const

driver

=

await

Driver.findOne({

user\_id:

user,

driver\_id:

driverId

});

if

(!driver)

{

return

NextResponse.json({

message:

'Driver

not

found'

},

{

status:

404

});

}

//

Find

the

index

of

the

account

with

the

specified

accountId

const

index

=

driver.accounts.findIndex((acc:

any)

=>

acc.account\_id

===

accountId);

if

(index

===

-1)

{

return

NextResponse.json({

message:

'Account

not

found'

},

{

status:

404

});

}

//

Update

the

account

with

the

new

data

console.log(driver.accounts[index])

driver.accounts[index].reason

=

data.reason

driver.accounts[index].gave

=

data.gave

driver.accounts[index].got

=

data.got

driver.accounts[index].date

=

data.date

//

Save

the

updated

driver

document

await

driver.save();

return

NextResponse.json({

driver

},

{

status:

200

});

}

catch

(error)

{

console.error('Failed

to

update

account

detail:',

error);

return

NextResponse.json({

error:

'Failed

to

update

account

detail'

},

{

status:

500

});

}

}

# drivers/[driverId]/calculateBalance

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

driverSchema,

ExpenseSchema,

tripSchema

}

from

"@/utils/schema";

import

{

models,

model

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema);

const

Driver

=

models.Driver

||

model('Driver',

driverSchema);

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

driverId:

string

}

})

{

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error

});

}

const

{

driverId

}

=

params;

await

connectToDatabase();

//

Aggregate

trips

with

accounts

const

accounts

=

await

Trip.aggregate([

{

$match:

{

user\_id:

user,

driver:

driverId

}

},

{

$unwind:

"$accounts"

},

{

$sort:

{

"dates.0":

-1

}

},

{

$project:

{

\_id:

0,

account:

"$accounts",

tripId:

"$trip\_id"

}

},

{

$replaceRoot:

{

newRoot:

{

$mergeObjects:

["$account",

{

tripId:

"$tripId"

}]

}

}

},

{

$match:

{

receivedByDriver:

true

}

}

//

Optional:

filter

accounts

received

by

driver

]);

//

Sum

the

amounts

from

expenses

const

expenseSum

=

await

Expense.aggregate([

{

$match:

{

user\_id:

user,

driver:

driverId

}

},

{

$group:

{

\_id:

null,

totalExpense:

{

$sum:

"$amount"

}

}

}

]);

//

Find

the

driver

and

calculate

driver

accounts

balance

const

driver

=

await

Driver.findOne({

user\_id:

user,

driver\_id:

driverId

}).select('accounts')

let

totalExpense

=

expenseSum.length

>

0

?

expenseSum[0].totalExpense

:

0;

let

totalAccounts

=

accounts.reduce((sum,

account)

=>

sum

+

account.amount,

0);

let

totalDriverAccounts

=

driver.accounts.reduce((sum

:

any,

account

:

any)

=>

sum

+

account.got

-

account.gave,

0);

const

total

=

totalDriverAccounts

+

totalAccounts

-

totalExpense;

return

NextResponse.json({

total

});

}

catch

(error

:

any)

{

return

NextResponse.json({

error:

error.message,

status:

500

});

}

}

# drivers/[driverId]/expense

## route.ts

import

{

verifyToken

}

from

"@/utils/auth"

import

{

connectToDatabase,

ExpenseSchema

}

from

"@/utils/schema"

import

{

model,

models

}

from

"mongoose"

import

{

NextResponse

}

from

"next/server"

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema)

export

async

function

GET(req

:

Request,

{params}

:

{params

:

{driverId

:

string}}){

const

{driverId}

=

params

try{

const

{user,

error}

=

await

verifyToken(req)

if(!user

||

error){

return

NextResponse.json({error})

}

await

connectToDatabase()

const

driverExpenses

=

await

Expense.find({user\_id

:

user,

driver

:

driverId}).lean()

return

NextResponse.json({driverExpenses,

status

:

200})

}catch(error){

return

NextResponse.json({error,

status

:

500})

}

}

# drivers/[driverId]/name

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

driverSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Driver

=

models.Driver

||

model('Driver',

driverSchema)

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

driverId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

driverId

}

=

params;

try

{

await

connectToDatabase();

const

driver

=

await

Driver.findOne({user\_id

:

user,

driver\_id

:

driverId}).select('name');

if

(!driver)

{

return

NextResponse.json({

message:

'Driver

not

found'

},

{

status:

404

});

}

return

NextResponse.json(driver,

{

status:

200

});

}

catch

(err:

any)

{

console.log(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

# parties

## route.ts

import

{

NextResponse,

NextRequest

}

from

'next/server';

import

{

model,

models

}

from

'mongoose';

import

{

connectToDatabase,

partySchema

}

from

'@/utils/schema';

import

{

IParty

}

from

'@/utils/interface';

import

{

auth

}

from

'@/firebase/firebaseAdmin';

import

{

fetchCookie,

verifyToken

}

from

'@/utils/auth';

const

Party

=

models.Party

||

model('Party',

partySchema);

export

async

function

GET(req

:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

parties

=

await

Party.find({user\_id

:

user}).lean().exec();

return

NextResponse.json({

parties

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

export

async

function

POST(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

data

=

await

req.json();

//

Basic

validation

//

GST

number

validation

const

gstRegex

=

/^[0-9]{2}[A-Z]{5}[0-9]{4}[A-Z]{1}[1-9A-Z]{1}Z[0-9A-Z]{1}$/;

if

(data.gstNumber

&&

!gstRegex.test(data.gstNumber))

{

return

NextResponse.json({

message:

'Invalid

GST

number'

},

{

status:

400

});

}

//

Phone

number

validation

(10

digits

starting

with

7,

8,

or

9)

const

phoneRegex

=

/^[789]\d{9}$/;

if

(data.contactNumber!=''

&&

!phoneRegex.test(data.contactNumber))

{

return

NextResponse.json({

message:

'Invalid

phone

number'

},

{

status:

400

});

}

const

newParty:

IParty

=

new

Party({

user\_id

:

user,

party\_id:

data.party\_id,

name:

data.name,

contactPerson:

data.contactPerson,

contactNumber:

data.contactNumber,

address:

data.address,

gstNumber:

data.gstNumber,

balance:

data.balance,

createdAt:

data.createdAt

||

new

Date(),

updatedAt:

data.updatedAt

||

new

Date(),

});

const

savedParty

=

await

newParty.save();

return

NextResponse.json({

message:

'Saved

Successfully',

data:

savedParty

},

{

status:

200

});

}

catch

(error:

any)

{

console.error('Error

saving

party:',

error);

if

(error.name

===

'ValidationError')

{

return

NextResponse.json({

message:

'Validation

Error',

details:

error.message

},

{

status:

400

});

}

else

if

(error.name

===

'MongoError'

&&

error.code

===

11000)

{

return

NextResponse.json({

message:

'Duplicate

Key

Error',

details:

error.message

},

{

status:

409

});

}

else

{

return

NextResponse.json({

message:

'Internal

Server

Error',

details:

error.message

},

{

status:

500

});

}

}

}

# parties/create

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

partySchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Party

=

models.Party

||

model('Party',

partySchema)

export

async

function

GET(req

:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

parties

=

await

Party.find({user\_id

:

user}).select(['name',

'party\_id']).lean().exec();

return

NextResponse.json({

parties

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# parties/[partyId]

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase

}

from

"@/utils/schema";

import

{

partySchema

}

from

"@/utils/schema";

import

{

models,

model

}

from

'mongoose'

import

{

NextResponse

}

from

"next/server";

const

Party

=

models.Party

||

model('Party',

partySchema)

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

partyId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

partyId

}

=

params;

try

{

await

connectToDatabase();

const

party

=

await

Party.findOne({

party\_id:

partyId,

user\_id:

user

}).lean().exec();

if

(!party)

{

return

NextResponse.json({

message:

'Party

not

found'

},

{

status:

404

});

}

return

NextResponse.json({

party:

party

},

{

status:

200

});

}

catch

(err:

any)

{

console.error(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

export

async

function

PUT(req:

Request,

{

params

}:

{

params:

{

partyId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

partyId

}

=

params;

const

data

=

await

req.json()

try

{

await

connectToDatabase();

const

gstRegex

=

/^[0-9]{2}[A-Z]{5}[0-9]{4}[A-Z]{1}[1-9A-Z]{1}Z[0-9A-Z]{1}$/;

if

(data.gstNumber

&&

!gstRegex.test(data.gstNumber))

{

return

NextResponse.json({

message:

'Invalid

GST

number'

},

{

status:

400

});

}

//

Phone

number

validation

(10

digits

starting

with

7,

8,

or

9)

const

phoneRegex

=

/^[789]\d{9}$/;

if

(data.contactNumber

!=

''

&&

!phoneRegex.test(data.contactNumber))

{

return

NextResponse.json({

message:

'Invalid

phone

number'

},

{

status:

400

});

}

const

party

=

await

Party.findOneAndUpdate({

party\_id:

partyId,

user\_id:

user

},

data,

{

new:

true

}).lean().exec();

if

(!party)

{

return

NextResponse.json({

message:

'Party

not

found'

},

{

status:

404

});

}

return

NextResponse.json({

party:

party

},

{

status:

200

});

}

catch

(err:

any)

{

console.error(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

export

async

function

DELETE(req:

Request,

{

params

}:

{

params:

{

partyId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

partyId

}

=

params;

try

{

await

connectToDatabase();

const

party

=

await

Party.findOneAndDelete({

party\_id:

partyId,

user\_id:

user

}).lean().exec();

if

(!party)

{

return

NextResponse.json({

message:

'Party

not

found'

},

{

status:

404

});

}

return

NextResponse.json({

party:

party

},

{

status:

200

});

}

catch

(err:

any)

{

console.error(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

# parties/[partyId]/calculateBalance

## route.ts

import

{

fetchBalanceBack

}

from

"@/helpers/fetchTripBalance";

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripChargesSchema,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

const

TripCharges

=

models.TripCharges

||

model('TripCharges',

tripChargesSchema);

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

partyId:

string

}

})

{

const

{

partyId

}

=

params;

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error,

status:

401

});

}

await

connectToDatabase();

const

trips

=

await

Trip.find({

user\_id:

user,

party:

partyId

}).select(['amount',

'accounts',

'trip\_id']);

let

totalbalance

=

0;

for

(const

trip

of

trips)

{

const

charges

=

await

TripCharges.find({

user\_id:

user,

trip\_id:

trip.trip\_id

});

totalbalance

+=

await

fetchBalanceBack(trip,

charges);

}

return

NextResponse.json({

totalbalance

});

}

catch

(error)

{

console.error(error);

return

NextResponse.json({

error:

'Internal

Server

Error',

status:

500

});

}

}

# suppliers

## route.ts

import

{

NextResponse,

NextRequest

}

from

'next/server';

import

mongoose,

{

model,

models

}

from

'mongoose';

import

{

connectToDatabase,

supplierSchema

}

from

'@/utils/schema';

import

{

ISupplier

}

from

'@/utils/interface';

import

{

v4

as

uuidv4

}

from

'uuid';

import

{

verifyToken

}

from

'@/utils/auth';

const

Supplier

=

models.Supplier

||

model('Supplier',

supplierSchema);

export

async

function

GET(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

suppliers

=

await

Supplier.find({

user\_id:

user

}).lean().exec();

return

NextResponse.json({

suppliers

},

{

status:

200

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

export

async

function

POST(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase

const

data

=

await

req.json();

//

Basic

validation

if

(!data.name)

{

return

NextResponse.json({

message:

'Missing

required

fields'

},

{

status:

400

});

}

const

phoneRegex

=

/^[789]\d{9}$/;

if

(data.contactNumber

!=

''

&&

!phoneRegex.test(data.contactNumber))

{

return

NextResponse.json({

message:

'Invalid

phone

number'

},

{

status:

400

});

}

const

newSupplier:

ISupplier

=

new

Supplier({

user\_id:

user,

supplier\_id:

'suppllier'

+

uuidv4(),

name:

data.name,

contactNumber:

data.contactNumber,

tripCount:

0,

balance:

0,

});

const

savedSupplier

=

await

newSupplier.save();

return

NextResponse.json({

message:

'Saved

Successfully',

data:

newSupplier

},

{

status:

200

});

}

catch

(error:

any)

{

console.error('Error

saving

party:',

error);

if

(error.name

===

'ValidationError')

{

return

NextResponse.json({

message:

'Validation

Error',

details:

error.message

},

{

status:

400

});

}

else

if

(error.name

===

'MongoError'

&&

error.code

===

11000)

{

return

NextResponse.json({

message:

'Duplicate

Key

Error',

details:

error.message

},

{

status:

409

});

}

else

{

return

NextResponse.json({

message:

'Internal

Server

Error',

details:

error.message

},

{

status:

500

});

}

}

}

# suppliers/[supplierId]

## route.ts

//

Import

necessary

modules

and

schemas

import

{

connectToDatabase,

supplierAccountSchema

}

from

"@/utils/schema";

import

{

models,

model

}

from

'mongoose';

import

{

supplierSchema

}

from

"@/utils/schema";

import

{

NextResponse

}

from

"next/server";

import

{

verifyToken

}

from

"@/utils/auth";

//

Retrieve

or

define

Mongoose

model

for

Supplier

const

Supplier

=

models.Supplier

||

model('Supplier',

supplierSchema);

const

SupplierAccount

=

models.SupplierAccount

||

model('SupplierAccount',

supplierAccountSchema)

//

GET request handler function export async function GET(req: Request, {params}: { params: { supplierId: string } }) { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } const { supplierId } = params; try { // Connect to the MongoDB database await connectToDatabase(); // Find the supplier based on supplierId const supplier = await Supplier.findOne({ user\_id: user, supplier\_id: supplierId }).lean(); // Handle case where supplier is not found if (!supplier) { return NextResponse.json({ message: "No supplier found" }, { status: 404 }); } // Return the supplier details return NextResponse.json({ supplier: supplier }, { status: 200 }); } catch (error) { console.error('Error fetching supplier:', error); return NextResponse.json({ message: "Internal Server Error" }, { status: 500 }); } } export async function PATCH(req: Request,{ params}: { params: { supplierId: String } }) { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } const { supplierId } = params; const data = await req.json() const newBalance = data.truckHireCost try { await connectToDatabase() const supplier = await Supplier.findOne({ user\_id: user, supplier\_id: supplierId }) supplier.balance = parseFloat(supplier.balance) + parseFloat(newBalance) await supplier.save() return NextResponse.json({ message: 'Balance updated successfully', balance: supplier.balance }, { status: 200 }) } catch (error) { console.log(error) return NextResponse.json({ message: "error" }, { status: 500 }) } } export async function DELETE(req: Request, {params}: { params: { supplierId: string } }) { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } const { supplierId } = params; try { // Connect to the MongoDB database await connectToDatabase(); // Find the supplier based on supplierId const supplier = await Supplier.findOneAndDelete({ user\_id: user, supplier\_id: supplierId }).lean(); await SupplierAccount.deleteMany({user\_id : user, supplier\_id : supplierId}) // Handle case where supplier is not found if (!supplier) { return NextResponse.json({ message: "No supplier found" }, { status: 404 }); } // Return the supplier details return NextResponse.json({ supplier: supplier }, { status: 200 }); } catch (error) { console.error('Error fetching supplier:', error); return NextResponse.json({ message: "Internal Server Error" }, { status: 500 }); } } export async function PUT(req: Request, {params}: { params: { supplierId: string } }) { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } const { supplierId } = params; const data = await req.json() try { // Connect to the MongoDB database await connectToDatabase(); // Find the supplier based on supplierId const phoneRegex = /^[789]\d{9}$/; if (data.contactNumber!='' && !phoneRegex.test(data.contactNumber)) { return NextResponse.json({ message: 'Invalid phone number' }, { status: 400 }); } const supplier = await Supplier.findOneAndUpdate({ user\_id: user, supplier\_id: supplierId }, data).lean(); // Handle case where supplier is not found if (!supplier) { return NextResponse.json({ message: "No supplier found" }, { status: 404 }); } // Return the supplier details return NextResponse.json({ supplier: supplier }, { status: 200 }); } catch (error) { console.error('Error fetching supplier:', error); return NextResponse.json({ message: "Internal Server Error" }, { status: 500 }); } }

# suppliers/[supplierId]/calculateBalance

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

supplierAccountSchema,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

const

SupplierAccount

=

models.SupplierAccount

||

model('SupplierAccount',

supplierAccountSchema);

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

supplierId:

string

}

})

{

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error

},

{

status:

401

});

}

const

{

supplierId

}

=

params;

await

connectToDatabase();

const

[supplierAccountSummary,

tripSummary]

=

await

Promise.all([

SupplierAccount.aggregate([

{

$match:

{

user\_id:

user,

supplier\_id:

supplierId

}

},

{

$group:

{

\_id:

null,

totalAmount:

{

$sum:

"$amount"

}

}

}

]),

Trip.aggregate([

{

$match:

{

user\_id:

user,

supplier:

supplierId

}

},

{

$group:

{

\_id:

null,

totalTruckHireCost:

{

$sum:

"$truckHireCost"

}

}

}

])

]);

const

totalAccountBalance

=

supplierAccountSummary[0]?.totalAmount

||

0;

const

totalTruckHireCost

=

tripSummary[0]?.totalTruckHireCost

||

0;

const

balance

=

totalAccountBalance

-

totalTruckHireCost;

return

NextResponse.json({

balance,

status:

200

});

}

catch

(error)

{

console.error("Error

fetching

data:",

error);

return

NextResponse.json({

error:

"Internal

Server

Error"

},

{

status:

500

});

}

}

# suppliers/[supplierId]/payments

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

supplierAccountSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

SupplierAccount

=

models.SupplierAccount

||

model('SupplierAccount',

supplierAccountSchema)

export

async

function

POST(req:

Request,

{

params

}:

{

params:

{

supplierId:

string

}

})

{

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error

});

}

const

{

supplierId

}

=

params;

const

payments

=

await

req.json();

//

Assuming

the

body

is

an

array

of

payments

await

connectToDatabase();

const

savedPayments

=

[];

for

(const

payment

of

payments)

{

const

supplierAccount

=

new

SupplierAccount({

user\_id:

user,

//

Assuming

user

contains

an

id

property

supplier\_id:

supplierId,

...payment

});

const

savedPayment

=

await

supplierAccount.save();

savedPayments.push(savedPayment);

}

return

NextResponse.json({

success:

true,

payments:

savedPayments

});

}

catch

(error:

any)

{

console.error('Error

saving

supplier

account:',

error);

return

NextResponse.json({

error:

error.message

||

'Internal

server

error'

});

}

}

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

supplierId:

string

}

})

{

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error

});

}

const

{

supplierId

}

=

params;

await

connectToDatabase();

const

supplierAccounts

=

await

SupplierAccount.find({user\_id

:

user,

supplier\_id

:

supplierId}).lean()

return

NextResponse.json({

status

:

200,

supplierAccounts

});

}

catch

(error:

any)

{

console.error('Error

saving

supplier

account:',

error);

return

NextResponse.json({

error:

error.message

||

'Internal

server

error'

});

}

}

# suppliers/[supplierId]/payments/trips

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema)

export

async

function

GET(req

:

Request,

{params}

:

{params

:

{supplierId

:

string}})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{supplierId}

=

params

try

{

await

connectToDatabase();

const

trips

=

await

Trip.find({user\_id

:

user,supplier

:

supplierId}).select(['trip\_id','route','truckHireCost','startDate','truck']).lean().sort({

'dates.0':

-1

}).exec();

return

NextResponse.json({

trips

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# suppliers/[supplierId]/payments/trips/[tripId]

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

supplierAccountSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

SupplierAccount

=

models.SupplierAccount

||

model('SupplierAccount',

supplierAccountSchema)

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

supplierId:

string,

tripId:

string

}

})

{

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error

});

}

const

{

supplierId,

tripId

}

=

params;

await

connectToDatabase();

const

supplierAccounts

=

await

SupplierAccount.find({

user\_id:

user,

supplier\_id:

supplierId,

trip\_id:

tripId

}).select('amount').lean();

const

totalAmount

=

supplierAccounts.reduce((sum,

account)

=>

sum

+

account.amount,

0);

return

NextResponse.json({

status:

200,

totalAmount

});

}

catch

(error:

any)

{

console.error('Error

saving

supplier

account:',

error);

return

NextResponse.json({

error:

error.message

||

'Internal

server

error'

});

}

}

# suppliers/[supplierId]/payments/[paymentId]

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

supplierAccountSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

SupplierAccount

=

models.SupplierAccount

||

model('SupplierAccount',

supplierAccountSchema);

export

async

function

DELETE(req:

Request,

{

params

}:

{

params:

{

paymentId:

string

}

})

{

try

{

//

Verify

the

token

and

user

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error:

'Unauthorized',

status:

401

});

}

//

Extract

the

paymentId

from

params

const

{

paymentId

}

=

params;

//

Connect

to

the

database

await

connectToDatabase();

//

Find

and

delete

the

SupplierAccount

by

paymentId

const

account

=

await

SupplierAccount.findByIdAndDelete(paymentId);

//

Check

if

the

account

was

found

and

deleted

if

(!account)

{

return

NextResponse.json({

error:

'Payment

not

found',

status:

404

});

}

//

Return

the

deleted

account

with

a

success

status

return

NextResponse.json({

deletedAccount:

account,

status:

200

});

}

catch

(error)

{

console.error('Error

deleting

payment:',

error);

//

Return

an

internal

server

error

status

return

NextResponse.json({

error:

'Internal

Server

Error',

status:

500

});

}

}

# suppliers/[supplierId]/tripCount

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema)

export

async

function

GET(req

:

Request,

{params}

:

{params

:

{supplierId

:

string}}){

try

{

const

{user,

error}

=

await

verifyToken(req)

if(!user

||

error){

return

NextResponse.json({error})

}

const

{supplierId}

=

params

await

connectToDatabase()

const

tripCount

=

await

Trip.countDocuments({user\_id

:

user,

supplier

:

supplierId})

return

NextResponse.json({tripCount,

status

:

200})

}

catch

(error)

{

return

NextResponse.json({error,

status

:

500})

}

}

# tripCharges/[chargeId]

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripChargesSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

TripCharges

=

models.TripCharges

||

model('TripCharges',

tripChargesSchema)

export

async

function

DELETE(req:

Request,

{

params

}:

{

params:

{

chargeId:

string

}

})

{

await

connectToDatabase();

const

{chargeId}

=

params

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error

});

}

const

expense

=

await

TripCharges.findByIdAndDelete(chargeId);

if

(!expense)

{

return

NextResponse.json({

message:

'Expense

not

found'

},

{

status:

404

});

}

return

NextResponse.json({

status:

200,

charge:

expense

});

}

catch

(error)

{

return

NextResponse.json({

message:

'An

error

occurred

while

deleting

the

expense'

},

{

status:

500

});

}

}

# tripExpense

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripChargesSchema,

ExpenseSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

TripCharges

=

models.TripCharges

||

model('TripCharges',

tripChargesSchema);

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema);

export

async

function

GET(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

url

=

new

URL(req.url);

const

month

=

url.searchParams.get('month');

const

year

=

url.searchParams.get('year');

console.log(`${month}

${year}`);

await

connectToDatabase();

if

(!month

||

!year)

{

const

tripExpense

=

await

Expense.find({

user\_id:

user

}).lean();

return

NextResponse.json({

tripExpense,

status:

200

});

}

const

monthMap:

{

[key:

string]:

number

}

=

{

January:

0,

February:

1,

March:

2,

April:

3,

May:

4,

June:

5,

July:

6,

August:

7,

September:

8,

October:

9,

November:

10,

December:

11

};

const

monthNumber

=

monthMap[month];

if

(monthNumber

===

undefined)

{

return

NextResponse.json({

error:

'Invalid

month

name',

status:

400

});

}

const

startDate

=

new

Date(parseInt(year),

monthNumber,

1);

console.log('Start

Date

:

'

+

startDate);

const

endDate

=

new

Date(parseInt(year),

monthNumber

+

1,

1);

console.log('End

Date

:

'

+

endDate);

try

{

const

[tripExpense,

truckExpense]

=

await

Promise.all([

TripCharges.find({

user\_id:

user,

date:

{

$gte:

startDate,

$lt:

endDate

},

partyBill:

false

}).lean(),

Expense.find({

user\_id:

user,

date:

{

$gte:

startDate,

$lt:

endDate

},

trip\_id:

{

$exists:

true,

$ne:

''

}

}).lean()

]);

const

combinedExpenses

=

[...tripExpense,

...truckExpense];

return

NextResponse.json({

combinedExpenses,

status:

200

});

}

catch

(err:

any)

{

console.log(err);

return

NextResponse.json({

message:

err.message,

status:

500

});

}

}

# tripExpense/calculate

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripChargesSchema,

ExpenseSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

TripCharges

=

models.TripCharges

||

model('TripCharges',

tripChargesSchema);

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema);

export

async

function

GET(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

},

{

status:

401

});

//

Unauthorized

}

const

url

=

new

URL(req.url);

const

month

=

url.searchParams.get('month');

const

year

=

url.searchParams.get('year');

await

connectToDatabase();

if

(!month

||

!year)

{

const

tripExpense

=

await

Expense.find({

user\_id:

user

}).lean();

//

Calculate

total

expense

const

totalExpense

=

tripExpense.reduce((sum,

expense)

=>

sum

+

(expense.amount

||

0),

0);

return

NextResponse.json({

tripExpense,

totalExpense,

status:

200

});

}

const

monthMap:

{

[key:

string]:

number

}

=

{

January:

0,

February:

1,

March:

2,

April:

3,

May:

4,

June:

5,

July:

6,

August:

7,

September:

8,

October:

9,

November:

10,

December:

11

};

const

monthNumber

=

monthMap[month];

if

(monthNumber

===

undefined)

{

return

NextResponse.json({

error:

'Invalid

month

name',

status:

400

});

}

const

startDate

=

new

Date(parseInt(year),

monthNumber,

1);

const

endDate

=

new

Date(parseInt(year),

monthNumber

+

1,

1);

try

{

const

[tripExpense,

truckExpense]

=

await

Promise.all([

TripCharges.find({

user\_id:

user,

date:

{

$gte:

startDate,

$lt:

endDate

},

partyBill:

false

}).lean(),

Expense.find({

user\_id:

user,

date:

{

$gte:

startDate,

$lt:

endDate

},

trip\_id:

{

$exists:

true,

$ne:

''

}

}).lean()

]);

//

Combine

expenses

and

calculate

the

total

const

combinedExpenses

=

[...tripExpense,

...truckExpense];

const

totalExpense

=

combinedExpenses.reduce((sum,

expense)

=>

sum

+

(expense.amount

||

0),

0);

return

NextResponse.json({

totalExpense,

status:

200

});

}

catch

(err:

any)

{

console.error('Error

fetching

expenses:',

err);

return

NextResponse.json({

message:

'Internal

Server

Error',

status:

500

});

}

}

# trips

## route.ts

import

{

NextResponse

}

from

'next/server';

import

mongoose,

{

model,

models

}

from

'mongoose';

import

{

tripSchema

}

from

'@/utils/schema';

import

{

connectToDatabase

}

from

'@/utils/schema';

import

{

ITrip

}

from

'@/utils/interface';

import

{v4

as

uuidv4}

from

'uuid'

import

{

partySchema

}

from

'@/utils/schema';

import

{

verifyToken

}

from

'@/utils/auth';

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

const

Party

=

models.Party

||

model('Party',

partySchema)

export

async

function

GET(req

:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase();

const

trips

=

await

Trip.find({user\_id

:

user}).lean().sort({

'dates.0':

-1

}).exec();

return

NextResponse.json({

trips

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

export

async

function

POST(this:

any,

req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase();

//

Establish

database

connection

const

data

=

await

req.json();

//

Parse

JSON

data

from

request

body

//

Basic

validation

(you

can

implement

your

own

validation

logic

here

if

needed)

const

datearr

=

[new

Date(data.startDate),

null,

null,

null,

null]

//

Create

a

new

Trip

instance

based

on

ITrip

interface

const

newTrip:

ITrip

=

new

Trip({

user\_id

:

user,

trip\_id:

'trip'

+

uuidv4(),

party:

data.party,

truck:

data.truck,

driver:

data.driver,

supplier

:

data.supplierId,

route:

{

origin:

data.route.origin,

destination:

data.route.destination

},

billingType:

data.billingType,

amount:

data.amount,

balance:

data.amount,

dates:

datearr,

//

Assuming

startDate

is

passed

as

string

and

needs

conversion

truckHireCost:

data.truckHireCost

||

0,

LR:

data.LR,

status

:

0,

material:

data.material

||

'',

notes:

data.notes

||

'',

accounts

:

[]

});

//

Save

the

new

trip

document

const

savedTrip

=

await

newTrip.save();

const

party

=

await

Party.findOne({party\_id

:

data.party})

party.balance

=

parseFloat(party.balance)

+

newTrip.amount

await

party.save()

//

Return

success

response

return

NextResponse.json({

message:

'Saved

Successfully',

data:

savedTrip

},

{

status:

200

});

}

catch

(error:

any)

{

console.error('Error

saving

trip:',

error);

//

Handle

different

types

of

errors

if

(error.name

===

'ValidationError')

{

return

NextResponse.json({

message:

'Validation

Error',

details:

error.message

},

{

status:

400

});

}

else

if

(error.name

===

'MongoError'

&&

error.code

===

11000)

{

return

NextResponse.json({

message:

'Duplicate

Key

Error',

details:

error.message

},

{

status:

409

});

}

else

{

return

NextResponse.json({

message:

'Internal

Server

Error',

details:

error.message

},

{

status:

500

});

}

}

}

# trips/driver/[driverId]/accounts

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

driverId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

const

{

driverId

}

=

params;

await

connectToDatabase();

const

accounts

=

await

Trip.aggregate([

{

$match:

{

user\_id:

user,

driver:

driverId

}

},

{

$unwind:

"$accounts"

},

{

$sort:

{

"dates.0":

-1

}

},

{

$project:

{

\_id:

0,

account:

"$accounts",

tripId:

"$trip\_id"

}},

{

$replaceRoot:

{

newRoot:

{

$mergeObjects:

["$account",

{

tripId:

"$tripId"

}]

}

}

},

{

$match:

{

receivedByDriver:

true

}

}

//

Optional:

filter

accounts

received

by

driver

]);

return

NextResponse.json({

accounts

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# trips/party/[partyId]

## route.ts

import

{

NextResponse

}

from

'next/server';

import

mongoose

from

'mongoose';

import

{

connectToDatabase,

tripSchema,

partySchema

}

from

'@/utils/schema';

import

{

verifyToken

}

from

'@/utils/auth';

const

Trip

=

mongoose.models.Trip

||

mongoose.model('Trip',

tripSchema);

const

Party

=

mongoose.models.Party

||

mongoose.model('Party',

partySchema);

export

async

function

GET(request:

Request,

{

params

}:

{

params:

{

partyId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(request);

if

(error)

{

return

NextResponse.json({

error

},

{

status:

401

});

}

try

{

await

connectToDatabase();

const

{

partyId

}

=

params;

const

trips

=

await

Trip.find({

user\_id:

user,

party:

partyId

})

.select(['LR',

'startDate',

'truck',

'route',

'status',

'accounts',

'trip\_id',

'amount'])

.sort({

'dates.0':

-1

})

//

Ensure

sorting

by

the

correct

field

.lean()

.exec();

return

NextResponse.json({

trips

});

}

catch

(err)

{

console.error('Error

fetching

trips:',

err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# trips/party/[partyId]/accounts

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

driverId:

string,

partyId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

const

{

partyId

}

=

params;

await

connectToDatabase();

const

accounts

=

await

Trip.aggregate([

{

$match:

{

user\_id:

user,

party:

partyId

}

},

{

$unwind:

"$accounts"

},

{

$sort:

{

"dates.0":

-1

}

},

{

$project:

{

\_id:

0,

account:

"$accounts",

trip\_id:

"$trip\_id"

}},

{

$replaceRoot:

{

newRoot:

{

$mergeObjects:

["$account",

{

trip\_id:

"$trip\_id"

}]

}

}

},//

Optional:

filter

accounts

received

by

driver

]);

return

NextResponse.json({

accounts

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# trips/supplier/[supplierId]

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

supplierSchema,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Supplier

=

models.Supplier

||

model('Supplier',

supplierSchema)

const

Trip

=

models.Trip

||

model('Trip',

tripSchema)

export

async

function

GET(req:

Request,

params:

{

params:

{

supplierId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

supplierId

}

=

params.params;

try

{

//

Connect

to

the

MongoDB

database

await

connectToDatabase();

//

Find

the

supplier

based

on

supplierId

const

trips

=

await

Trip.find({user\_id

:

user,

supplier

:

supplierId}).select(['trip\_id','startDate','truck','route','truckHireCost','status',]).lean()

//

Handle

case

where

supplier

is

not

found

if

(!trips)

{

return

NextResponse.json({

message:

"No

trips

found"

},

{

status:

404

});

}

//

Return

the

supplier

details

return

NextResponse.json({

trips

:

trips

},

{

status:

200

});

}

catch

(error)

{

console.error('Error

fetching

supplier:',

error);

return

NextResponse.json({

message:

"Internal

Server

Error"

},

{

status:

500

});

}

}

# trips/truck/[truckNo]

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema)

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

truckNo:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

const

{

truckNo

}

=

params;

await

connectToDatabase();

const

trips

=

await

Trip.find({

user\_id:

user,

truck:

truckNo

}).lean().sort({

'dates.0':

-1

}).exec();

return

NextResponse.json({

trips

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# trips/[tripId]

## route.ts

import

DriverModal

from

"@/components/driver/driverModal";

import

{

fetchBalance,

fetchBalanceBack

}

from

"@/helpers/fetchTripBalance";

import

{

verifyToken

}

from

"@/utils/auth";

import

{

ITrip,

PaymentBook

}

from

"@/utils/interface";

import

{

connectToDatabase,

driverSchema,

partySchema,

tripChargesSchema,

truckSchema

}

from

"@/utils/schema";

import

{

tripSchema

}

from

"@/utils/schema";

import

{

models,

model

}

from

'mongoose'

import

{

NextResponse

}

from

"next/server";

import

{

v4

as

uuidv4

}

from

"uuid";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema)

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

tripId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

tripId

}

=

params;

try

{

await

connectToDatabase();

const

trip

=

await

Trip.findOne({

user\_id:

user,

trip\_id:

tripId

}).lean().exec();

if

(!trip)

{

return

NextResponse.json({

message:

'Trip

not

found'

},

{

status:

404

});

}

return

NextResponse.json({

trip

},

{

status:

200

});

}

catch

(err:

any)

{

console.error(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

export

async

function

PATCH(req:

Request,

{

params

}:

{

params:

{

tripId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

const

{

tripId

}

=

params;

const

{

data

}

=

await

req.json();

const

{

amount,

POD,

status,

dates,

account,

notes

}

=

data;

await

connectToDatabase();

const

trip

=

await

Trip.findOne({

user\_id:

user,

trip\_id:

tripId

});

if

(!trip)

{

return

NextResponse.json({

message:

'No

Trip

Found'

},

{

status:

404

});

}

if

(notes)

{

trip.notes

=

notes

}

if

(account)

{

if

(account.paymentBook\_id)

{

trip.accounts

=

trip.accounts.filter((acc:

PaymentBook)

=>

acc.paymentBook\_id

=

account.paymentBook\_id)

trip.accounts.push(account)

}

else

{

account.paymentBook\_id

=

'payment'

+

uuidv4()

trip.accounts.push(account);

}

const

TripExpense

=

models.TripExpense

||

model('TripExpense',

tripChargesSchema)

const

charges

=

await

TripExpense.find({

user\_id:

user,

trip\_id:

trip.trip\_id

})

const

pending

=

await

fetchBalanceBack(trip,

charges)

if

(pending

<

0)

{

return

NextResponse.json({

message:

"Balance

going

negative",

status:

400

})

}

}

if

(status

&&

dates)

{

trip.status

=

status;

trip.dates

=

dates;

}

trip.POD

=

POD

||

"";

await

trip.save();

return

NextResponse.json({

trip:

trip

},

{

status:

200

});

}

catch

(err:

any)

{

console.log(err);

return

NextResponse.json({

message:

err.message

},

{

status:

500

});

}

}

export

async

function

PUT(req:

Request,

{

params

}:

{

params:

{

tripId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

tripId

}

=

params;

try

{

await

connectToDatabase();

//

Assuming

data

is

correctly

parsed

from

req.json()

const

{

data

}

=

await

req.json();

//

Assuming

models.Truck

and

models.Driver

are

defined

elsewhere

const

Truck

=

models.Truck

||

model('Truck',

truckSchema);

const

Driver

=

models.Driver

||

model('Driver',

driverSchema);

const

oldTrip

=

await

Trip.findOne({

user\_id:

user,

trip\_id:

tripId

})

const

TripExpense

=

models.TripExpense

||

model('TripExpense',

tripChargesSchema)

const

charges

=

await

TripExpense.find({

user\_id:

user,

trip\_id:

oldTrip.trip\_id

})

const

pending

=

await

fetchBalanceBack(oldTrip,charges)

if

(pending

<

0)

{

return

NextResponse.json({

message:

"Balance

going

negative",

status:

400

})

}

const

trip

=

await

Trip.findOneAndUpdate({

user\_id:

user,

trip\_id:

tripId

},

data,

{

new:

true

});

if

(!trip)

{

return

NextResponse.json({

message:

'Trip

not

found'

},

{

status:

404

});

}

//

Update

driver

status

to

'On

Trip'

await

Driver.findOneAndUpdate({

driver\_id:

trip.driver

},

{

status:

'On

Trip'

});

//

Update

truck

status

to

'On

Trip'

await

Truck.findOneAndUpdate({

truckNo:

trip.truck

},

{

status:

'On

Trip'

});

//

Return

updated

trip

with

status

200

return

NextResponse.json({

trip

},

{

status:

200

});

}

catch

(err:

any)

{

console.error(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

export

async

function

DELETE(req:

Request,

{

params

}:

{

params:

{

tripId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

tripId

}

=

params;

const

Truck

=

models.Truck

||

model('Truck',

truckSchema);

const

Driver

=

models.Driver

||

model('Driver',

driverSchema);

try

{

await

connectToDatabase();

const

trip

=

await

Trip.findOneAndDelete({

user\_id:

user,

trip\_id:

tripId

}).exec();

if

(!trip)

{

return

NextResponse.json({

message:

'Trip

not

found'

},

{

status:

404

});

}

await

Truck.findOneAndUpdate({

truckNo:

trip.truck

},

{

status:

'Available'

})

await

Driver.findOneAndUpdate({

driver\_id:

trip.driver

},

{

status:

'Available'

})

return

NextResponse.json({

trip

},

{

status:

200

});

}

catch

(err:

any)

{

console.error(err);

return

NextResponse.json({

message:

'Internal

Server

Error',

error:

err.message

},

{

status:

500

});

}

}

# trips/[tripId]/accounts/[accountId]

## route.ts

import

{

fetchBalance,

fetchBalanceBack

}

from

"@/helpers/fetchTripBalance";

import

{

verifyToken

}

from

"@/utils/auth";

import

{

ITrip

}

from

"@/utils/interface";

import

{

connectToDatabase,

tripChargesSchema,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

export

async

function

DELETE(req:

Request,

{

params

}:

{

params:

{

tripId:

string,

accountId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

tripId,

accountId

}

=

params;

await

connectToDatabase();

try

{

//

Fetch

the

trip

const

trip:

ITrip

|

null

=

await

Trip.findOne({

user\_id:

user,

trip\_id:

tripId

});

//

Check

if

trip

exists

if

(!trip)

{

return

NextResponse.json({

message:

'Trip

not

found'

},

{

status:

404

});

}

//

Filter

out

the

account

to

be

deleted

trip.accounts

=

trip.accounts.filter(acc

=>

acc.\_id.toString()

!==

accountId);

const

TripExpense

=

models.TripExpense

||

model('TripExpense',

tripChargesSchema)

const

charges

=

await

TripExpense.find({

user\_id:

user,

trip\_id:

trip.trip\_id

})

const

pending

=

await

fetchBalanceBack(trip,

charges)

if

(pending

<

0)

{

return

NextResponse.json({

message:

"Balance

going

negative",

status:

400

})

}

//

Save

the

updated

trip

await

trip.save();

//

Return

success

response

return

NextResponse.json({

trip:

trip

},

{

status:

200

});

}

catch

(error:

any)

{

console.error(error);

return

NextResponse.json({

message:

'Failed

to

delete

account',

error:

error.message

},

{

status:

500

});

}

}

export

async

function

PATCH(req:

Request,

{

params

}:

{

params:

{

tripId:

string,

accountId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

{

tripId,

accountId

}

=

params;

const

{

account

}

=

await

req.json()

await

connectToDatabase();

try

{

//

Fetch

the

trip

const

trip:

ITrip

|

null

=

await

Trip.findOne({

user\_id:

user,

trip\_id:

tripId

});

//

Check

if

trip

exists

if

(!trip)

{

return

NextResponse.json({

message:

'Trip

not

found'

},

{

status:

404

});

}

//

Filter

out

the

account

to

be

deleted

const

index

=

trip.accounts.findIndex(acc

=>

acc.\_id.toString()

===

accountId)

trip.accounts[index]

=

account

const

TripExpense

=

models.TripExpense

||

model('TripExpense',

tripChargesSchema)

const

charges

=

await

TripExpense.find({

user\_id:

user,

trip\_id:

trip.trip\_id

})

const

pending

=

await

fetchBalanceBack(trip,

charges)

if

(pending

<

0)

{

return

NextResponse.json({

message:

"Balance

going

negative",

status:

400

})

}

//

Save

the

updated

trip

await

trip.save();

//

Return

success

response

return

NextResponse.json({

trip:

trip

},

{

status:

200

});

}

catch

(error:

any)

{

console.error(error);

return

NextResponse.json({

message:

'Failed

to

delete

account',

error:

error.message

},

{

status:

500

});

}

}

# trips/[tripId]/expenses

## route.ts

import

{

fetchBalanceBack

}

from

"@/helpers/fetchTripBalance";

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripChargesSchema,

tripSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextRequest,

NextResponse

}

from

"next/server";

//

Initialize

the

TripExpense

model

const

TripCharges

=

models.TripCharges

||

model('TripCharges',

tripChargesSchema);

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

tripId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

//

Connect

to

the

database

await

connectToDatabase();

//

Extract

the

tripId

from

the

request

params

const

{

tripId

}

=

params;

try

{

//

Fetch

the

trip

expenses

from

the

database

const

charges

=

await

TripCharges.find({

user\_id:

user,

trip\_id:

tripId

}).lean();

//

Return

a

success

response

with

the

charges

return

NextResponse.json({

status:

200,

charges

});

}

catch

(error)

{

//

Handle

any

errors

that

occur

during

the

process

console.error("Error

fetching

trip

expenses:",

error);

return

NextResponse.json({

status:

500,

error:

"Failed

to

fetch

trip

expenses"

});

}

}

//

Define

the

POST handler export async function POST(req: Request, { params }: { params: { tripId: string } }) { // Connect to the database await connectToDatabase(); // Extract the tripId from the request params const { tripId } = params; try { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } // Parse the request body as JSON const data = await req.json(); // Create a new instance of TripExpense with the parsed data and tripId const newCharge = new TripCharges({ ...data, trip\_id: tripId, user\_id: user }); const Trip = models.Trip || model('Trip', tripSchema) const trip = await Trip.findOne({user\_id : user, trip\_id : tripId}) const charges = await TripCharges.find({ user\_id: user, trip\_id: trip.trip\_id }) const pending = await fetchBalanceBack(trip, charges) if (pending < 0) { return NextResponse.json({ message: "Balance going negative", status: 400 }) } // Save the new charge to the database await newCharge.save(); // Return a success response with the new charge return NextResponse.json({ status: 200, newCharge }); } catch (error) { // Handle any errors that occur during the process console.error("Error creating new trip expense:", error); return NextResponse.json({ status: 500, error: "Failed to create new trip expense" }); } } export async function PATCH(req: Request, { params }: { params: { tripId: string } }) { await connectToDatabase(); const edited = await req.json(); try { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } const expense = await TripCharges.findOneAndUpdate({user\_id : user, \_id : edited.\_id}, edited, { new: true }); if (!expense) { return NextResponse.json({ message: 'Expense not found' }, { status: 404 }); } return NextResponse.json({ status: 200, charge: expense }); } catch (error) { return NextResponse.json({ message: 'An error occurred while updating the expense' }, { status: 500 }); } } export async function DELETE(req: Request, { params }: { params: { tripId: string } }) { await connectToDatabase(); const { id } = await req.json() try { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } const expense = await TripCharges.findOneAndDelete({user\_id : user, \_id : id}); if (!expense) { return NextResponse.json({ message: 'Expense not found' }, { status: 404 }); } return NextResponse.json({ status: 200, charge: expense }); } catch (error) { return NextResponse.json({ message: 'An error occurred while deleting the expense' }, { status: 500 }); } }

# trips/[tripId]/truckExpense

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

tripSchema,

ExpenseSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema)

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

tripId:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

//

Connect

to

the

database

await

connectToDatabase();

//

Extract

the

tripId

from

the

request

params

const

{

tripId

}

=

params;

try

{

//

Fetch

the

trip

expenses

from

the

database

const

charges

=

await

Expense.find({

user\_id:

user,

trip\_id:

tripId

}).lean();

//

Return

a

success

response

with

the

charges

return

NextResponse.json({

status:

200,

charges

});

}

catch

(error)

{

//

Handle

any

errors

that

occur

during

the

process

console.error("Error

fetching

trip

expenses:",

error);

return

NextResponse.json({

status:

500,

error:

"Failed

to

fetch

expenses"

});

}

}

export

async

function

POST(req:

Request,

{

params

}:

{

params:

{

tripId:

string

}

})

{

//

Connect

to

the

database

await

connectToDatabase();

//

Extract

the

tripId

from

the

request

params

const

{

tripId

}

=

params;

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

//

Parse

the

request

body

as

JSON

const

data

=

await

req.json();

if

(data.partyBill

&&

data.expenseType

==

'Fuel

Expense')

{

return

NextResponse.json({

status:

400,

message:

"Cannot

Add

Fuel

to

Party

Bill"

})

}

//

Create

a

new

instance

of

TripExpense

with

the

parsed

data

and

tripId

const

charge

=

new

Expense({

...data,

trip\_id:

tripId,

user\_id:

user

});

//

Save

the

new

charge

to

the

database

await

charge.save();

//

Return

a

success

response

with

the

new

charge

return

NextResponse.json({

status:

200,

charge

});

}

catch

(error)

{

//

Handle

any

errors

that

occur

during

the

process

console.error("Error

creating

new

trip

expense:",

error);

return

NextResponse.json({

status:

500,

error:

"Failed

to

create

new

expense"

});

}

}

# truckExpense

## route.ts

import

{

useAuth

}

from

"@/components/AuthProvider";

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

ExpenseSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema)

export

async

function

GET(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

const

url

=

new

URL(req.url);

const

month

=

url.searchParams.get('month');

const

year

=

url.searchParams.get('year');

console.log(`${month}

${year}`);

if

(!month

||

!year)

{

await

connectToDatabase();

const

expenses

=

await

Expense.find({

user\_id:

user

});

return

NextResponse.json({

expenses,

status:

200

});

}

//

Map

of

month

names

to

month

numbers

(0-indexed)

const

monthMap:

{

[key:

string]:

number

}

=

{

January:

0,

February:

1,

March:

2,

April:

3,

May:

4,

June:

5,

July:

6,

August:

7,

September:

8,

October:

9,

November:

10,

December:

11

};

const

monthNumber

=

monthMap[month];

if

(monthNumber

===

undefined)

{

return

NextResponse.json({

error:

'Invalid

month

name',

status:

400

});

}

const

startDate

=

new

Date(year

as

any,

monthNumber,

1);

console.log('Start

Date

:

'

+

startDate)

const

endDate

=

new

Date(year

as

any,

monthNumber

+

1,

1);

//

Next

month's

start

date

console.log('End

Date

:

'

+

endDate)

try

{

await

connectToDatabase();

const

expenses

=

await

Expense.find({

user\_id:

user,

date:

{

$gte:

startDate,

$lt:

endDate,

},

$or:

[

{

trip\_id:

{

$exists:

false

}

},

{

trip\_id:

{

$eq:

''

}

}

]

}).lean();

return

NextResponse.json({

truckExpense

:

expenses,

status:

200

});

}

catch

(err:

any)

{

console.log(err);

return

NextResponse.json({

message:

err.message,

status:

500

});

}

}

export

async

function

POST(req:

Request)

{

//

Connect

to

the

database

await

connectToDatabase();

//

Extract

the

tripId

from

the

request

params

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

//

Parse

the

request

body

as

JSON

const

data

=

await

req.json();

console.log(data)

//

Create

a

new

instance

of

TripExpense

with

the

parsed

data

and

tripId

const

newCharge

=

new

Expense({

...data,

user\_id:

user

});

//

Save

the

new

charge

to

the

database

await

newCharge.save();

//

Return

a

success

response

with

the

new

charge

return

NextResponse.json({

status:

200,

newCharge

});

}

catch

(error)

{

//

Handle

any

errors

that

occur

during

the

process

console.error("Error

creating

new

trip

expense:",

error);

return

NextResponse.json({

status:

500,

error:

"Failed

to

create

new

expense"

});

}

}

# truckExpense/calculate

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

ExpenseSchema

}

from

"@/utils/schema";

import

{

model,

models

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

//

Define

the

Expense

model

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema);

export

async

function

GET(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

},

{

status:

401

});

//

Unauthorized

}

const

url

=

new

URL(req.url);

const

month

=

url.searchParams.get('month');

const

year

=

url.searchParams.get('year');

if

(!month

||

!year)

{

await

connectToDatabase();

const

expenses

=

await

Expense.find({

user\_id:

user

}).lean();

//

Calculate

the

total

expense

const

totalExpense

=

expenses.reduce((sum,

expense)

=>

sum

+

(expense.amount

||

0),

0);

return

NextResponse.json({

expenses,

totalExpense,

status:

200

});

}

//

Map

of

month

names

to

month

numbers

(0-indexed)

const

monthMap:

{

[key:

string]:

number

}

=

{

January:

0,

February:

1,

March:

2,

April:

3,

May:

4,

June:

5,

July:

6,

August:

7,

September:

8,

October:

9,

November:

10,

December:

11

};

const

monthNumber

=

monthMap[month];

if

(monthNumber

===

undefined)

{

return

NextResponse.json({

error:

'Invalid

month

name',

status:

400

});

}

const

startDate

=

new

Date(Number(year),

monthNumber,

1);

const

endDate

=

new

Date(Number(year),

monthNumber

+

1,

1);

//

Next

month's

start

date

try

{

await

connectToDatabase();

const

expenses

=

await

Expense.find({

user\_id:

user,

date:

{

$gte:

startDate,

$lt:

endDate,

},

$or:

[

{

trip\_id:

{

$exists:

false

}

},

{

trip\_id:

{

$eq:

''

}

}

]

}).lean();

//

Calculate

the

total

expense

const

totalExpense

=

expenses.reduce((sum,

expense)

=>

sum

+

(expense.amount

||

0),

0);

return

NextResponse.json({totalExpense,

status:

200

});

}

catch

(err:

any)

{

console.error('Error

fetching

expenses:',

err);

return

NextResponse.json({

message:

'Internal

Server

Error',

status:

500

});

}

}

# trucks

## route.ts

import

{

NextResponse

}

from

'next/server';

import

mongoose,

{

model,

models

}

from

'mongoose';

import

{

connectToDatabase,

truckSchema

}

from

'@/utils/schema';

import

{

TruckModel

}

from

'@/utils/interface';

import

{

verifyToken

}

from

'@/utils/auth';

import

{v4

as

uuidv4}

from

'uuid'

const

Truck

=

models.Truck

||

model('Truck',

truckSchema);

export

async

function

GET(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

trucks

=

await

Truck.find({

user\_id:

user

}).exec();

return

NextResponse.json({

trucks

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

export

async

function

POST(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase();

//

Ensure

database

connection

is

established

//

Parse

incoming

JSON

data

from

request

body

const

data

=

await

req.json();

//

Validate

required

fields

if

(!data.truckNo)

{

throw

new

Error('Truck

Number

is

required');

}

if

(!data.ownership)

{

throw

new

Error('Ownership

type

is

required');

}

if

(data.ownership

===

'Market'

&&

!data.supplier)

{

throw

new

Error('Supplier

is

required

for

Market

ownership');

}

//

Create

a

new

TruckModel

instance

with

provided

data

const

newTruck

=

new

Truck({

user\_id:

user,

truck\_id

:

'truck\_id'

+

uuidv4(),

truckNo:

data.truckNo,

truckType:

data.truckType

||

'',

model:

data.model

||

'',

capacity:

data.capacity

||

'',

bodyLength:

data.bodyLength

||

null,

ownership:

data.ownership,

supplier:

data.supplier

||

'',

status:

'Available',

trip\_id:

'',

createdAt:

new

Date(),

updatedAt:

new

Date()

});

//

Save

the

new

truck

instance

to

the

database

const

truck

=

await

newTruck.save();

//

Return

successful

response

with

created

truck

data

return

NextResponse.json({

truck

},

{

status:

200

});

}

catch

(error:

any)

{

//

Handle

errors

during

request

processing

console.error('Error

creating

truck:',

error);

//

Return

error

response

with

appropriate

status

code

and

message

return

NextResponse.json({

error:

error.message

||

'Failed

to

create

truck'

},

{

status:

500

});

}

}

# trucks/create

## route.ts

import

{

NextResponse

}

from

'next/server';

import

{

model,

models

}

from

'mongoose';

import

{

connectToDatabase,

truckSchema

}

from

'@/utils/schema';

import

{

verifyToken

}

from

'@/utils/auth';

const

Truck

=

models.Truck

||

model('Truck',

truckSchema);

export

async

function

GET(req:

Request)

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

await

connectToDatabase()

const

trucks

=

await

Truck.find({

user\_id:

user

}).select(['truckNo','status','supplier']).exec();

return

NextResponse.json({

trucks

});

}

catch

(err)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error'

},

{

status:

500

});

}

}

# trucks/[truckNo]

## route.ts

import

{

NextResponse

}

from

'next/server';//

Ensure

to

import

Request

from

'express'

or

another

appropriate

package

import

{

connectToDatabase,

ExpenseSchema,

tripSchema,

truckSchema

}

from

'@/utils/schema';

import

{

TruckModel

}

from

'@/utils/interface';

import

mongoose,

{

model,

Model,

models

}

from

'mongoose';

import

{

verifyToken

}

from

'@/utils/auth';

const

Truck

=

mongoose.models.Truck

||

mongoose.model<TruckModel>('Truck',

truckSchema);

const

Trip

=

models.Trip

||

model('Trip',

tripSchema)

const

Expense

=

models.Expense

||

model('Expense',ExpenseSchema)

export

async

function

PATCH(req:

Request,

{

params

}:

{

params:

{

truckNo:

string

}

})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

try

{

const

{

truckNo

}

=

params;

const

{

status

}

=

await

req.json();

//

Assuming

'status'

is

in

the

body

of

the

PATCH request await connectToDatabase(); // Ensure this function is properly defined and imported const truck = await Truck.findOne({user\_id : user, truckNo: truckNo }); if (!truck) { return NextResponse.json({ message: 'No Truck Found' }, { status: 404 }); } if (status) truck.status = status; await truck.save(); return NextResponse.json({ truck: truck }, { status: 200 }); } catch (err: any) { return NextResponse.json({ message: err.message }, { status: 500 }); } } export async function PUT(req: Request, { params }: { params: { truckNo: string } }) { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }, { status: 401 }); } try { const { truckNo } = params; const data = await req.json(); await connectToDatabase(); const truck = await Truck.findOneAndUpdate({ user\_id: user, truckNo }, data, { new: true }); if (!truck) { return NextResponse.json({ message: 'No Truck Found' }, { status: 404 }); } const trips = await Trip.find({ user\_id: user, truck: truckNo }); await Promise.all(trips.map(async (trip) => { trip.truck = data.truckNo; // Update the truck number in each trip await trip.save(); })); // Update the truck number in the Expense collection const updatedExpenses = await Expense.updateMany( { user\_id: user, truck: truckNo }, // Query to find matching documents with the old truck number { $set: { truck: data.truckNo } } // Update operation to set the new truck number ); return NextResponse.json({ truck }, { status: 200 }); } catch (err: any) { return NextResponse.json({ message: err.message }, { status: 500 }); } } export async function GET(req: Request, { params }: { params: { truckNo: string } }) { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } try { const { truckNo } = params;// Assuming 'status' is in the body of the

PATCH request await connectToDatabase(); // Ensure this function is properly defined and imported const truck = await Truck.findOne({user\_id : user, truckNo: truckNo }); if (!truck) { return NextResponse.json({ message: 'No Truck Found' }, { status: 404 }); } return NextResponse.json({ truck: truck }, { status: 200 }); } catch (err: any) { return NextResponse.json({ message: err.message }, { status: 500 }); } } export async function DELETE(req: Request, { params }: { params: { truckNo: string } }) { const { user, error } = await verifyToken(req); if (error) { return NextResponse.json({ error }); } try { const { truckNo } = params;// Assuming 'status' is in the body of the

PATCH request await connectToDatabase(); // Ensure this function is properly defined and imported const foundTruck = await Truck.findOne({user\_id : user, truckNo: truckNo }); if(foundTruck.status == 'On Trip'){ return NextResponse.json({message : "Truck currently on Trip", status : 400}) } const truck = await Truck.findOneAndDelete({user\_id : user, truckNo: truckNo }); if (!truck) { return NextResponse.json({ message: 'No Truck Found' }, { status: 404 }); } return NextResponse.json({ truck: truck }, { status: 200 }); } catch (err: any) { return NextResponse.json({ message: err.message }, { status: 500 }); } }

# trucks/[truckNo]/expense

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

ExpenseSchema

}

from

"@/utils/schema";

import

{

NextResponse

}

from

"next/server";

import

{

maintenanceChargeTypes

}

from

"@/utils/utilArray";

import

{

model,

models

}

from

"mongoose";

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema)

export

async

function

GET(req:

Request,

{

params

}:

{

params:

{

truckNo:

string

}

})

{

const

{

truckNo

}

=

params;

const

url

=

new

URL(req.url);

const

expenseType

=

url.searchParams.get('type');

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

},

{

status:

401

});

}

await

connectToDatabase();

try

{

let

filter

:

any=

{

user\_id:

user,

truck:

truckNo

};

if

(expenseType

===

'fuel')

{

filter

=

{

...filter,

expenseType:

'Fuel

Expense'

};

}

else

if

(expenseType

===

'maintenance')

{

filter

=

{

...filter,

expenseType:

{

$in:

Array.from(maintenanceChargeTypes)

}

};

}

else

if

(expenseType

===

'other')

{

filter

=

{

...filter,

expenseType:

{

$nin:

Array.from(maintenanceChargeTypes)

}

};

}

const

expenses

=

await

Expense.find(filter).sort({

date:

-1

});

return

NextResponse.json(expenses);

}

catch

(error:

any)

{

console.log(error);

return

NextResponse.json({

message:

error.message,

status:

500

});

}

}

export

async

function

POST(req:

Request,

{

params

}:

{

params:

{

truckNo:

string

}

})

{

//

Connect

to

the

database

await

connectToDatabase();

//

Extract

the

tripId

from

the

request

params

const

{

truckNo

}

=

params;

try

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(error)

{

return

NextResponse.json({

error

});

}

//

Parse

the

request

body

as

JSON

const

data

=

await

req.json();

//

Create

a

new

instance

of

TripExpense

with

the

parsed

data

and

tripId

const

newCharge

=

new

Expense({

...data,

user\_id:

user

});

//

Save

the

new

charge

to

the

database

await

newCharge.save();

//

Return

a

success

response

with

the

new

charge

return

NextResponse.json({

status:

200,

newCharge

});

}

catch

(error)

{

//

Handle

any

errors

that

occur

during

the

process

console.error("Error

creating

new

trip

expense:",

error);

return

NextResponse.json({

status:

500,

error:

"Failed

to

create

new

trip

expense"

});

}

}

# trucks/[truckNo]/summary

## route.ts

import

{

verifyToken

}

from

"@/utils/auth";

import

{

connectToDatabase,

ExpenseSchema,

tripSchema,

tripChargesSchema

}

from

"@/utils/schema";

import

{

model,

models,

Schema

}

from

"mongoose";

import

{

NextResponse

}

from

"next/server";

//

Define

models

if

not

already

defined

const

Trip

=

models.Trip

||

model('Trip',

tripSchema);

const

Expense

=

models.Expense

||

model('Expense',

ExpenseSchema);

const

TripCharge

=

models.TripCharge

||

model('TripCharge',

tripChargesSchema);

export

async

function

GET(req:

Request,

{params}

:

{params

:

{truckNo

:

string}})

{

const

{

user,

error

}

=

await

verifyToken(req);

if

(!user

||

error)

{

return

NextResponse.json({

error,

status:

401

});

}

try

{

await

connectToDatabase();

const

{truckNo}

=

params

//

Fetch

all

trips

for

the

user

const

trips

=

await

Trip.find({

user\_id:

user

,

truck

:

truckNo});

//

Calculate

tripRevenue

let

tripRevenue

=

0;

for

(const

trip

of

trips)

{

tripRevenue

+=

trip.amount;

//

Fetch

trip

charges

for

the

current

trip

const

tripCharges

=

await

TripCharge.find({user\_id

:

user,

trip\_id:

trip.trip\_id

});

//

Calculate

the

charges

for

(const

charge

of

tripCharges)

{

if

(charge.partyBill)

{

tripRevenue

+=

charge.amount;

}

else

{

tripRevenue

-=

charge.amount;

}

}

}

//

Fetch

all

expenses

for

the

user

const

expenses

=

await

Expense.find({

user\_id:

user,

truck:

truckNo

}).select('amount');

//

Calculate

truckExpense

let

truckExpense

=

0;

for

(const

expense

of

expenses)

{

truckExpense

+=

expense.amount;

}

return

NextResponse.json({

truckExpense,

tripRevenue

},

{

status:

200

});

}

catch

(err:

any)

{

console.error(err);

return

NextResponse.json({

error:

'Internal

Server

Error',

status:

500

});

}

}