Daniel Gorman

Service Layers

IRP Quests

Overview

The backend services for my application will be built using Prisma ORM in NextJs. I have created a simple RESTful API which will be manipulated by the front end to store and change data.

The back end is broken up into three layers.

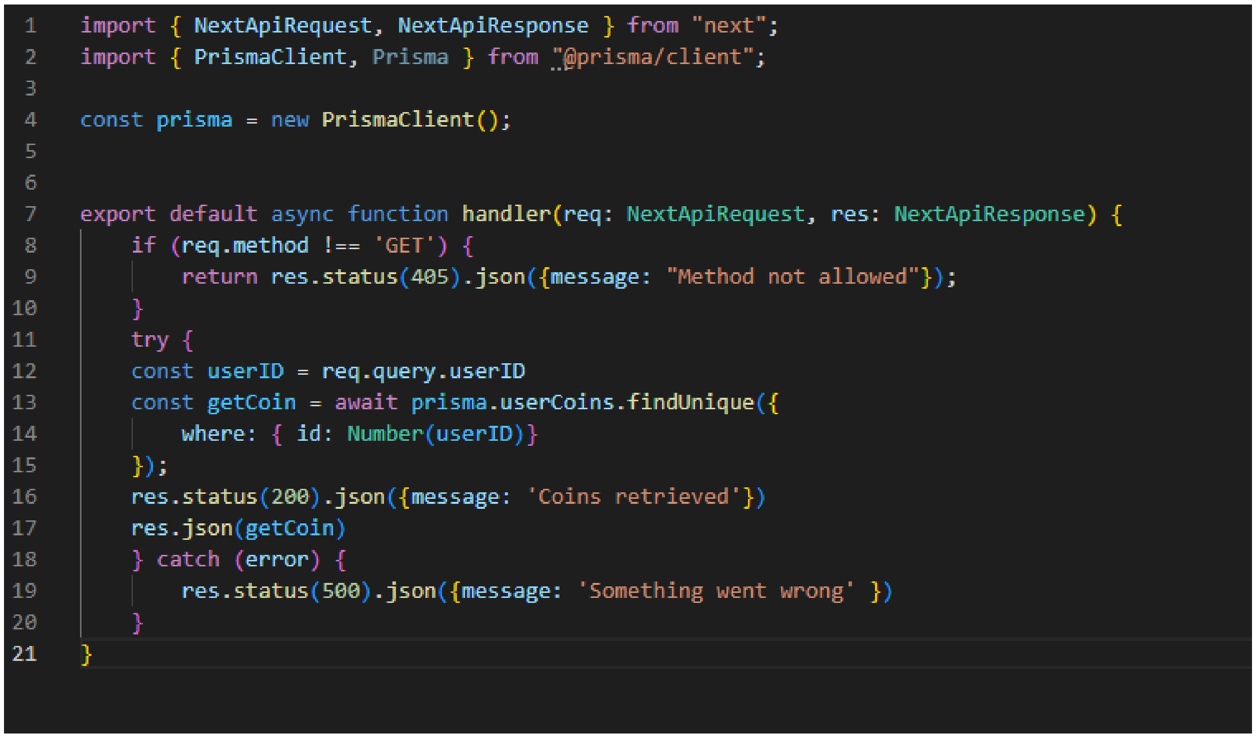
1. Route: this will define the paths that are available and can be used for specific information.

2. Controller: The controller will be handling incoming HTTP requests and returns a response.

3. Service: The controllers will call the service methods that have logic to handle the data that is being requested or sent. Methods on the layer will be used to handle the database and manipulate the data.

The Gets

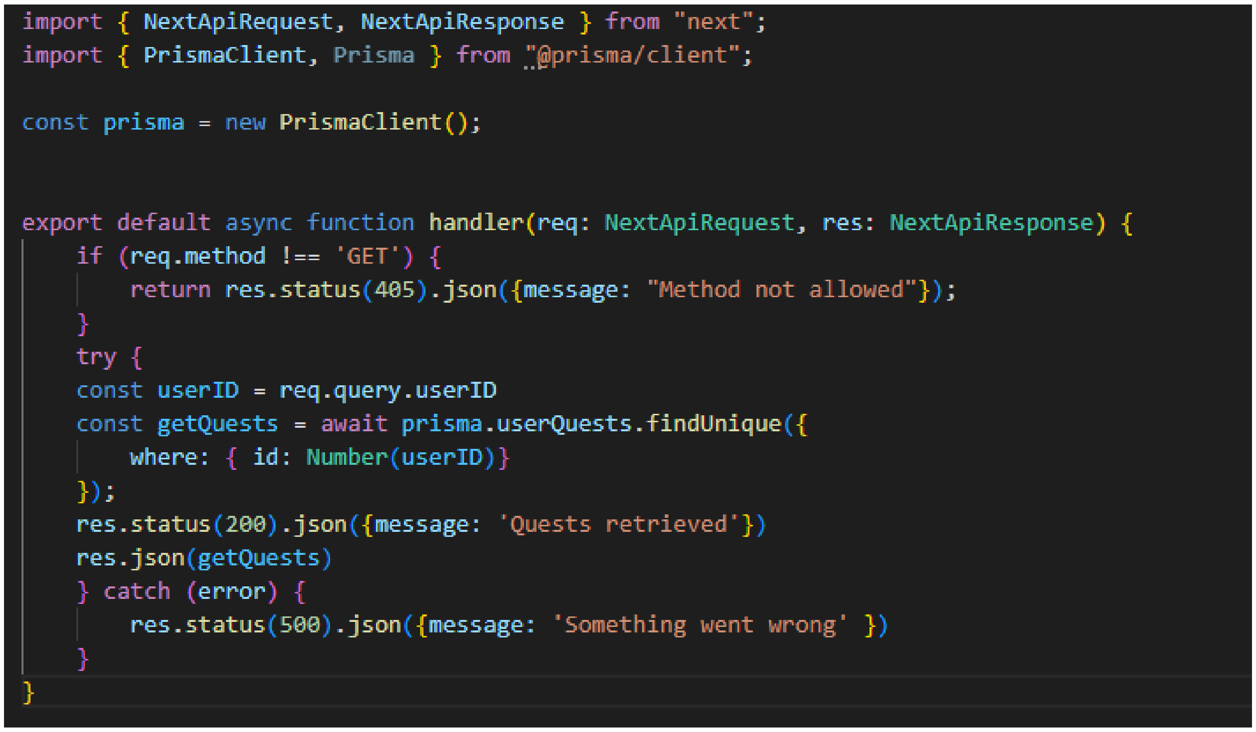
Get coin amount



```
import { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();


export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method !== 'GET') {
        return res.status(405).json({message: "Method not allowed"});
    }
    try {
    const userID = req.query.userID
    const getCoin = await prisma.userCoins.findUnique({
        where: { id: Number(userID)}
    });
    res.status(200).json({message: 'Coins retrieved'})
    res.json(getCoin)
    } catch (error) {
        res.status(500).json({message: 'Something went wrong' })
    }
}
```

This purpose is to get the amount of coins the user account contains based on their userID. This information is then used by the UI to display how many coins they can spend.

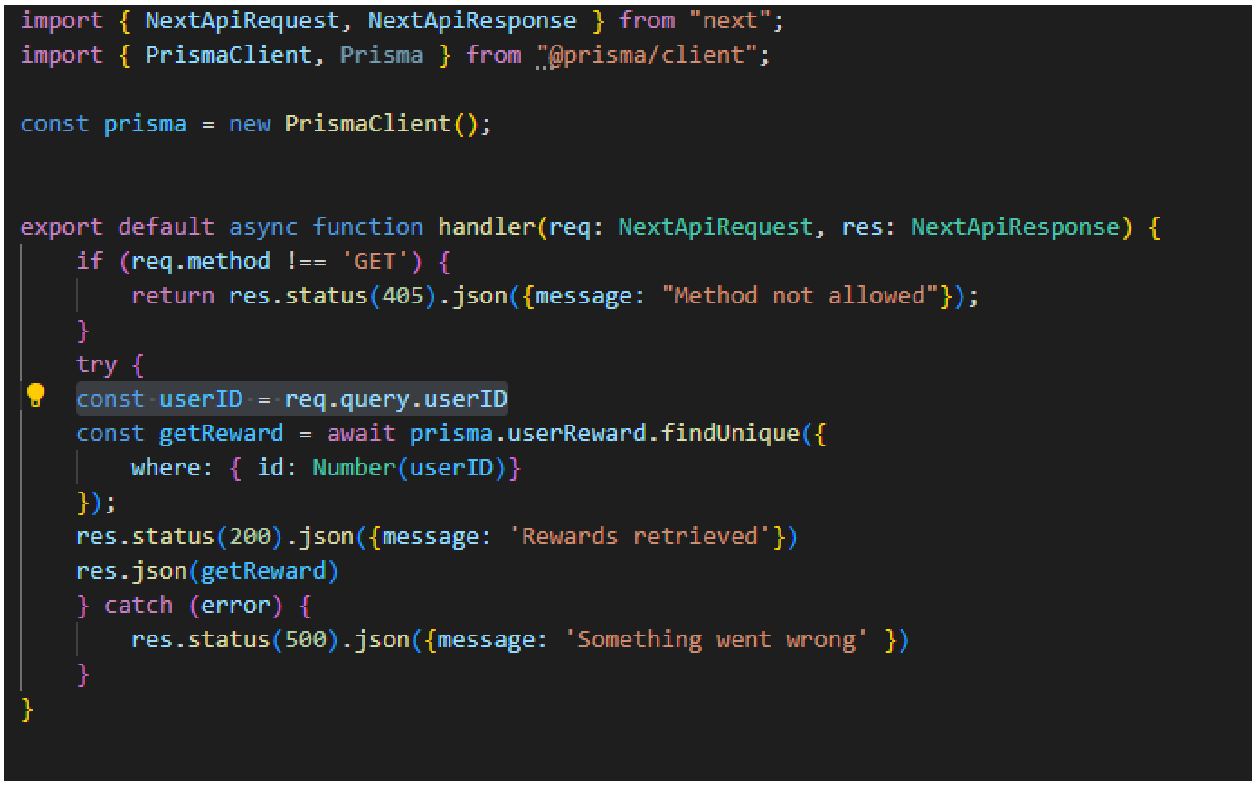
Get Quests



```
import { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();


export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method !== 'GET') {
        return res.status(405).json({message: "Method not allowed"});
    }
    try {
    const userID = req.query.userID
    const getQuests = await prisma.userQuests.findUnique({
        where: { id: Number(userID)}
    });
    res.status(200).json({message: 'Quests retrieved'})
    res.json(getQuests)
    } catch (error) {
        res.status(500).json({message: 'Something went wrong' })
    }
}
```

This purpose is to get the quests the user account contains based on their userID.
This information is then used by the UI to show the user the quests they have to complete, or have completed on their home screen.

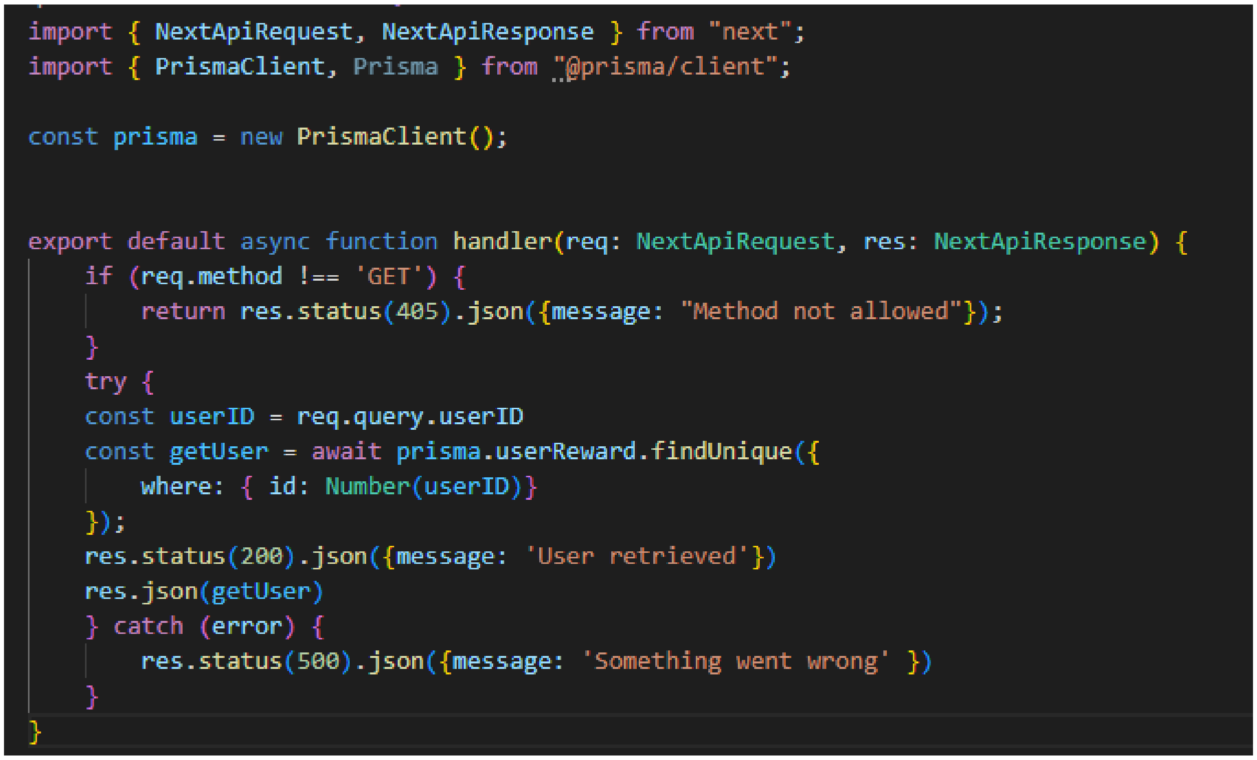
Get Rewards



```
import { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();


export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method !== 'GET') {
        return res.status(405).json({message: "Method not allowed"});
    }
    try {
    const userID = req.query.userID
    const getReward = await prisma.userReward.findUnique({
        where: { id: Number(userID)}
    });
    res.status(200).json({message: 'Rewards retrieved'})
    res.json(getReward)
    } catch (error) {
        res.status(500).json({message: 'Something went wrong' })
    }
}
```

The purpose is to get the Rewards so that the user knows what to spend their coins on,

Get user account

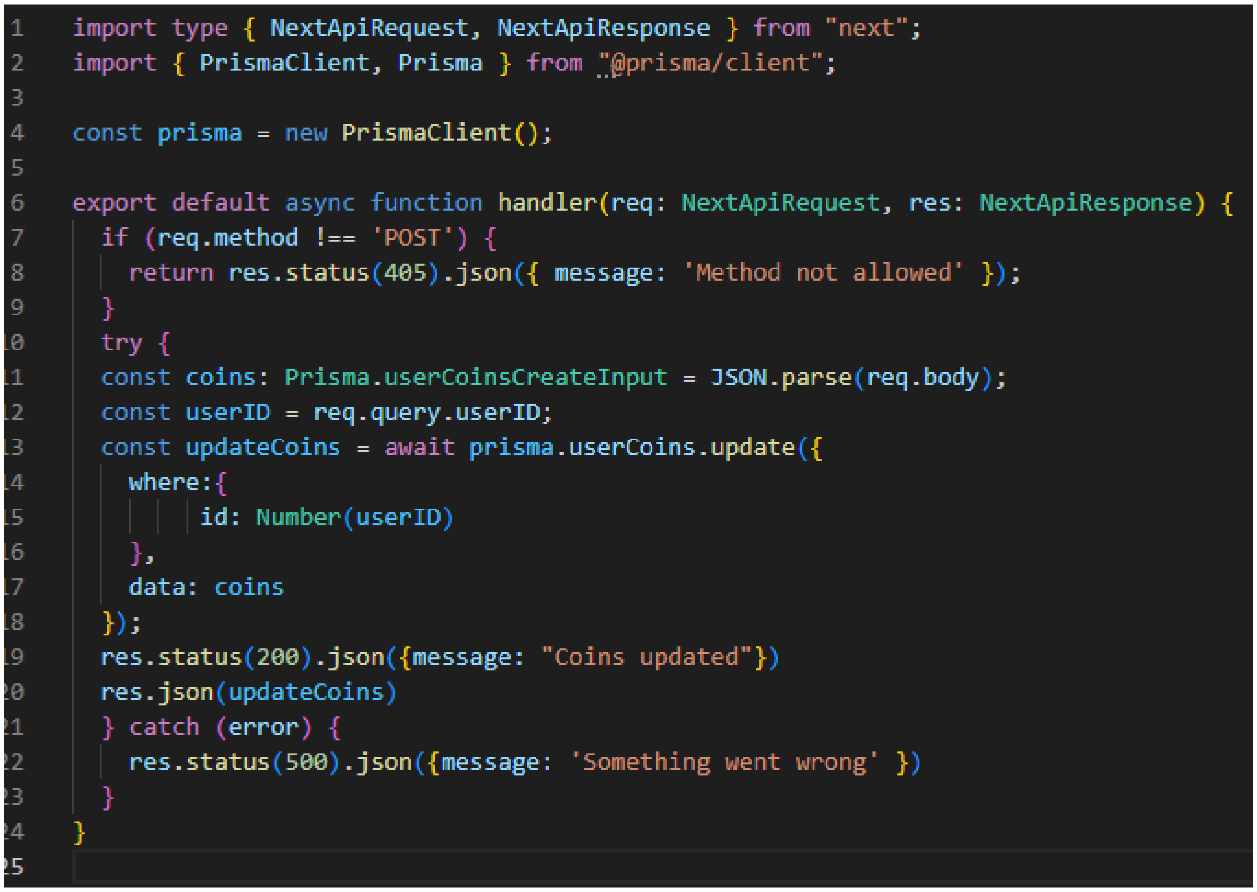


```
import { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();


export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method !== 'GET') {
        return res.status(405).json({message: "Method not allowed"});
    }
    try {
    const userID = req.query.userID
    const getUser = await prisma.userReward.findUnique({
        where: { id: Number(userID)}
    });
    res.status(200).json({message: 'User retrieved'})
    res.json(getUser)
    } catch (error) {
        res.status(500).json({message: 'Something went wrong' })
    }
}
```

The purpose is to get the details of the user’s account to display in the home screen or change the information as needed.

The Updates

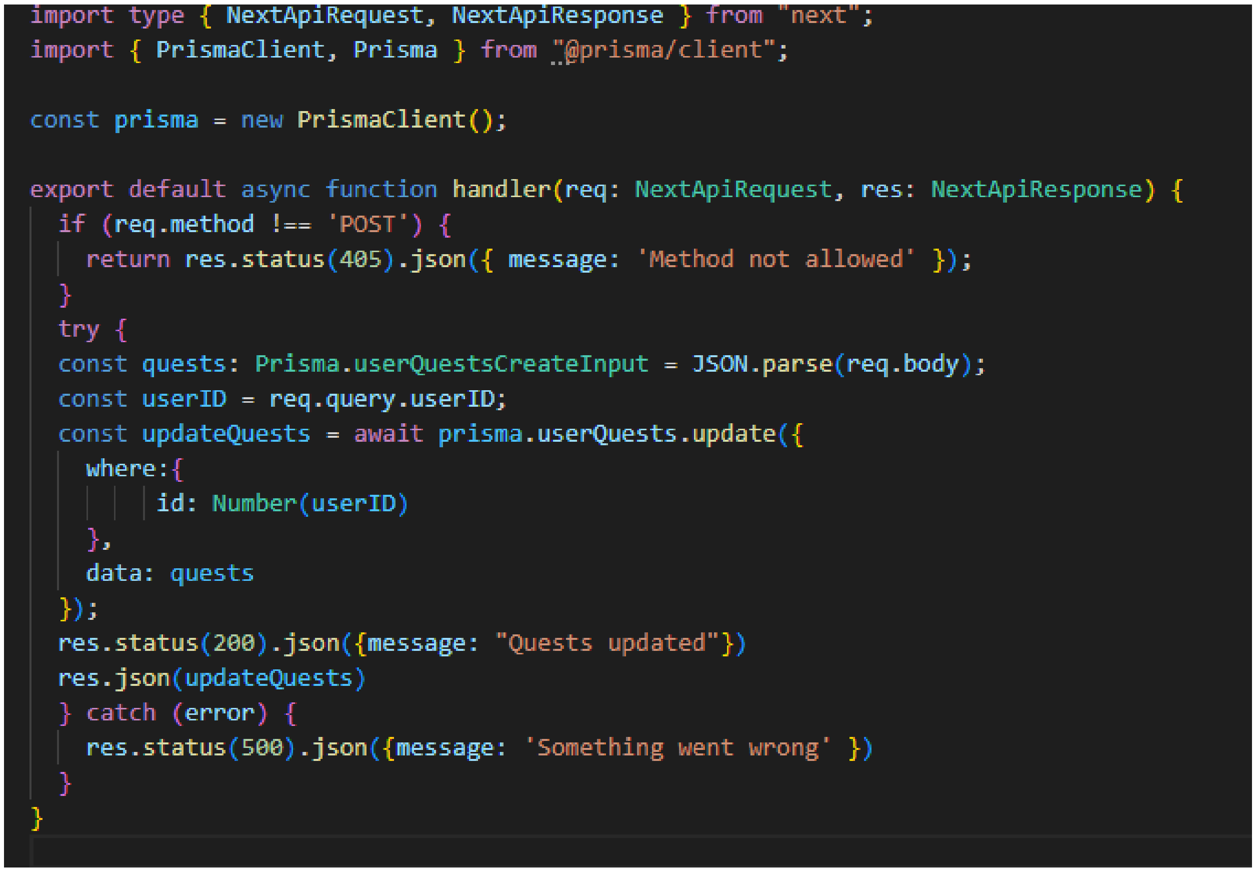
Update Coin Amount (Put)



```
import type { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  try {
  const coins: Prisma.userCoinsCreateInput = JSON.parse(req.body);
  const userID = req.query.userID;
  const updateCoins = await prisma.userCoins.update({
    where:{
        id: Number(userID)
    },
    data: coins
  });
  res.status(200).json({message: "Coins updated"})
  res.json(updateCoins)
  } catch (error) {
    res.status(500).json({message: 'Something went wrong' })
  }
}

```

The purpose of this is to allow for the update for the amount of coins the user has.

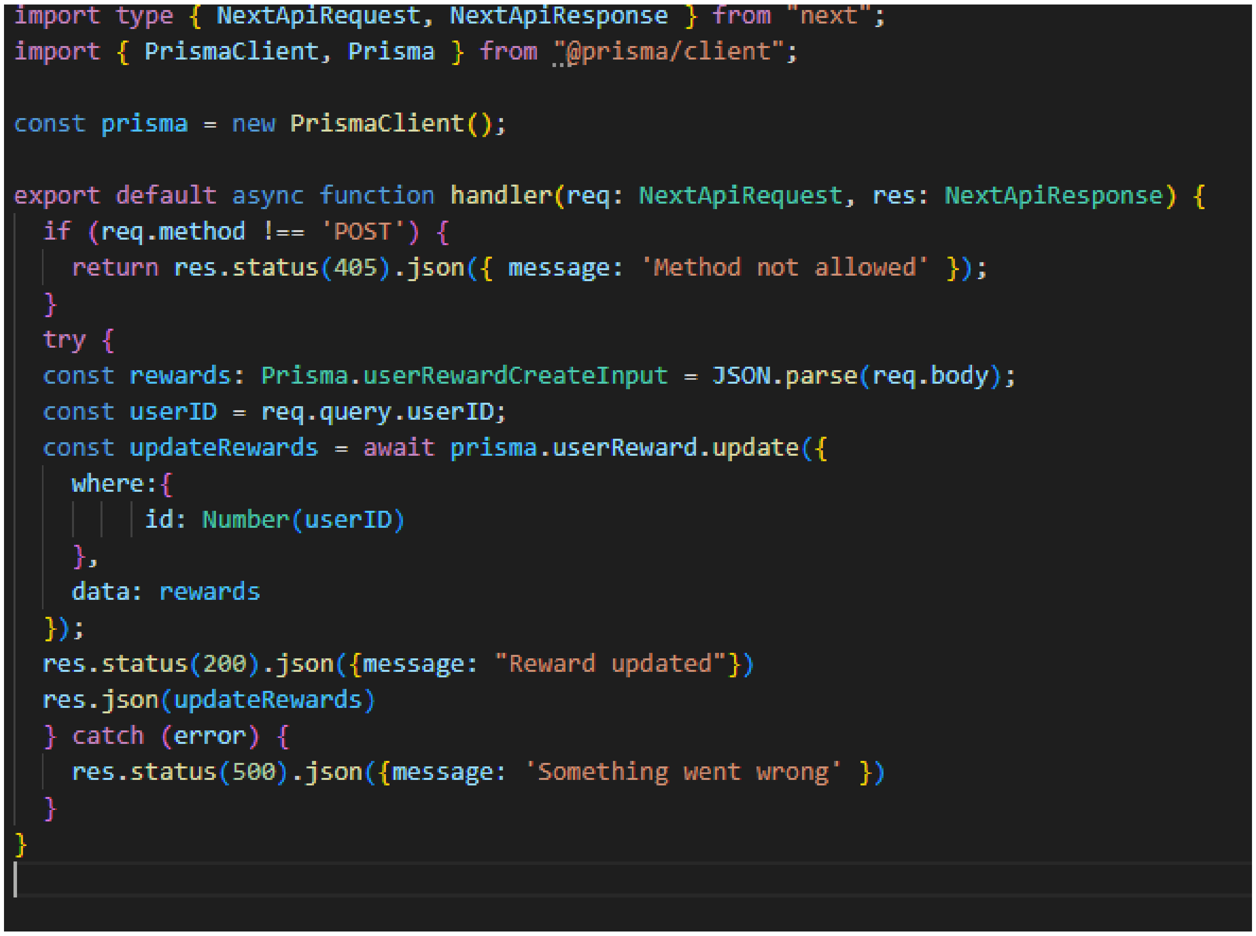
Update Quests (Put)



```
import type { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  try {
  const quests: Prisma.userQuestsCreateInput = JSON.parse(req.body);
  const userID = req.query.userID;
  const updateQuests = await prisma.userQuests.update({
    where:{
          id: Number(userID)
    },
    data: quests
  });
  res.status(200).json({message: "Quests updated"})
  res.json(updateQuests)
  } catch (error) {
    res.status(500).json({message: 'Something went wrong' })
  }
}
```

The purpose of this is to allow for the user to update their quests if they need a longer deadline or they add more objectives to the quest.

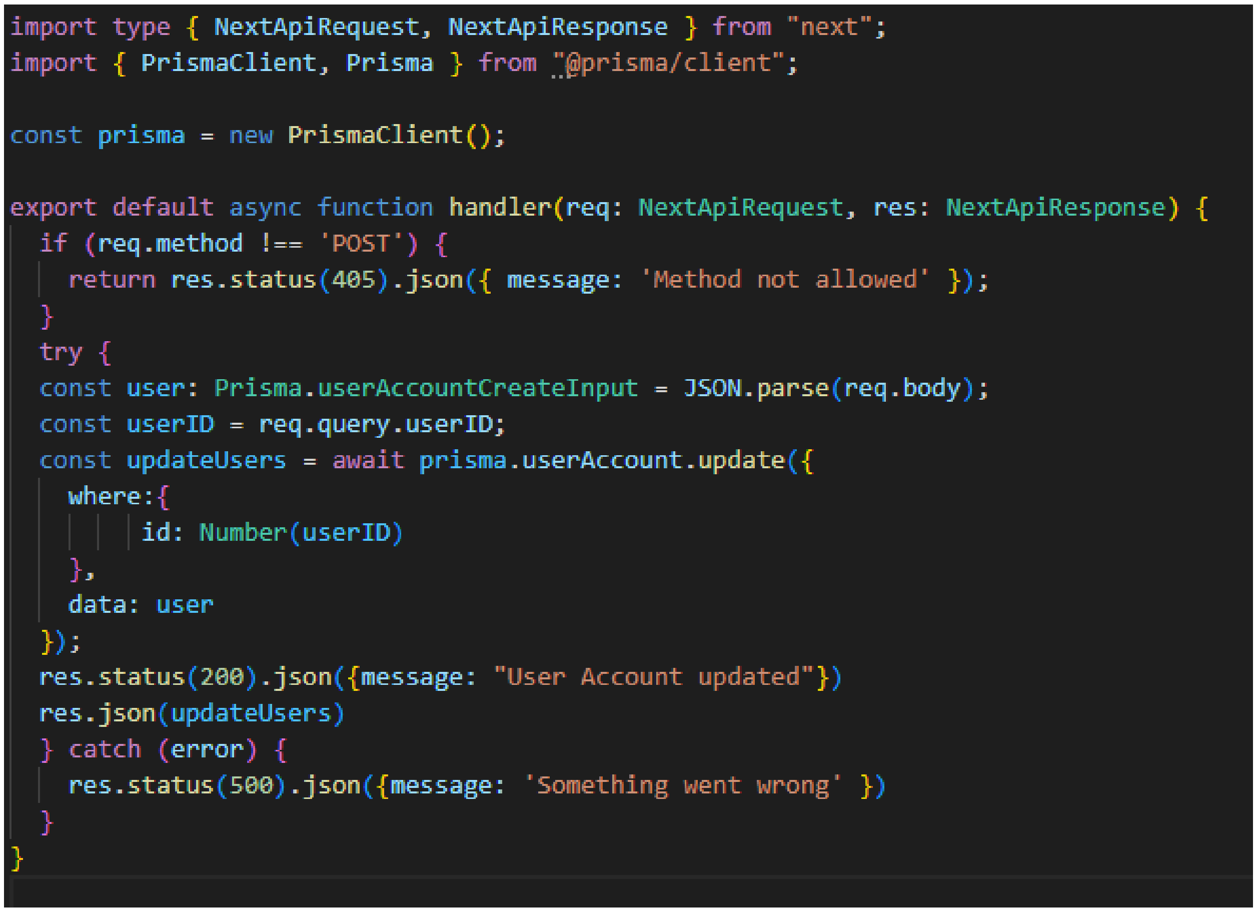
Update Rewards (Put)



```
import type { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  try {
  const rewards: Prisma.userRewardCreateInput = JSON.parse(req.body);
  const userID = req.query.userID;
  const updateRewards = await prisma.userReward.update({
    where:{
          id: Number(userID)
    },
    data: rewards
  });
  res.status(200).json({message: "Reward updated"})
  res.json(updateRewards)
  } catch (error) {
    res.status(500).json({message: 'Something went wrong' })
  }
}
```

The purpose of this is to allow the user to update their Rewards by either changing coin costs or what the reward entails.

Update user account (Put)

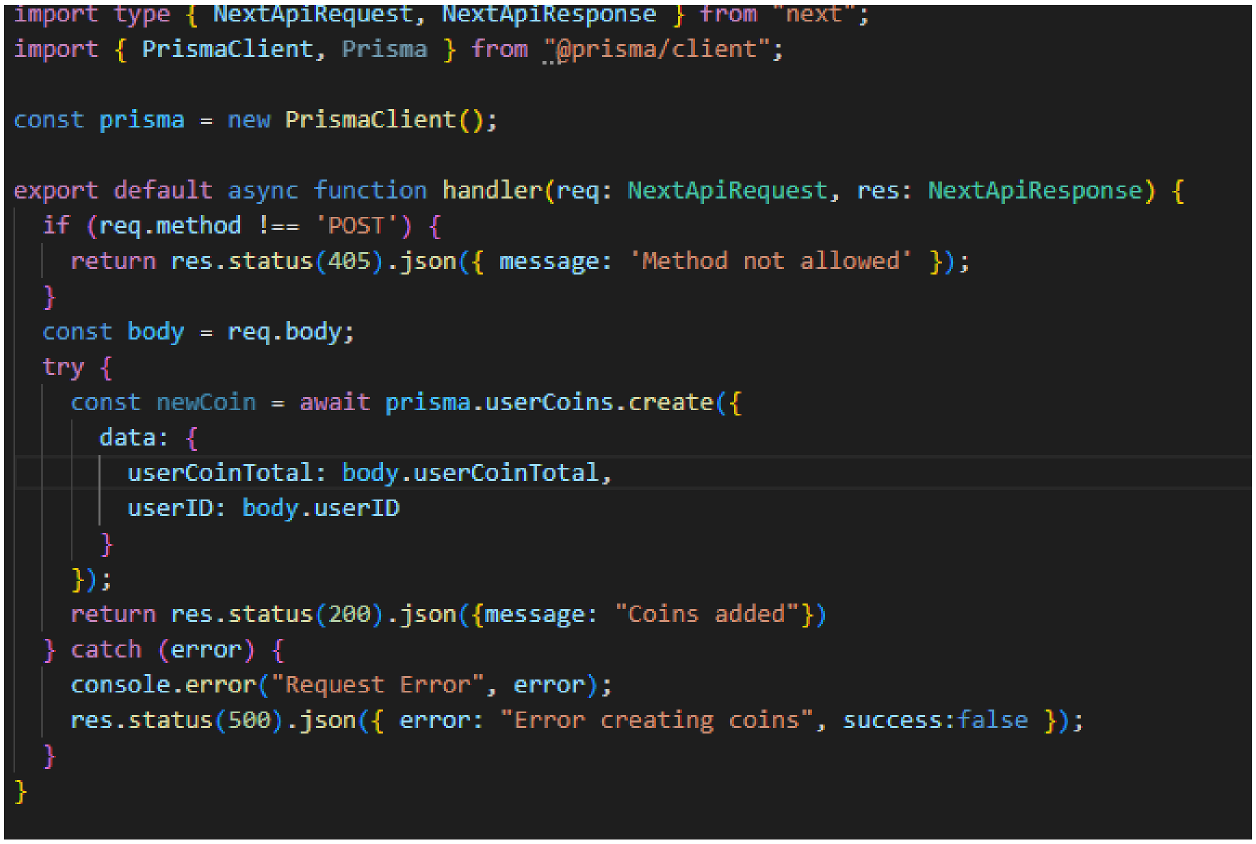


```
import type { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  try {
  const user: Prisma.userAccountCreateInput = JSON.parse(req.body);
  const userID = req.query.userID;
  const updateUsers = await prisma.userAccount.update({
    where:{
          id: Number(userID)
    },
    data: user
  });
  res.status(200).json({message: "User Account updated"})
  res.json(updateUsers)
  } catch (error) {
    res.status(500).json({message: 'Something went wrong' })
  }
}
```

The purpose of this is to allow for the user to update their password or email address as needed.

The Creates

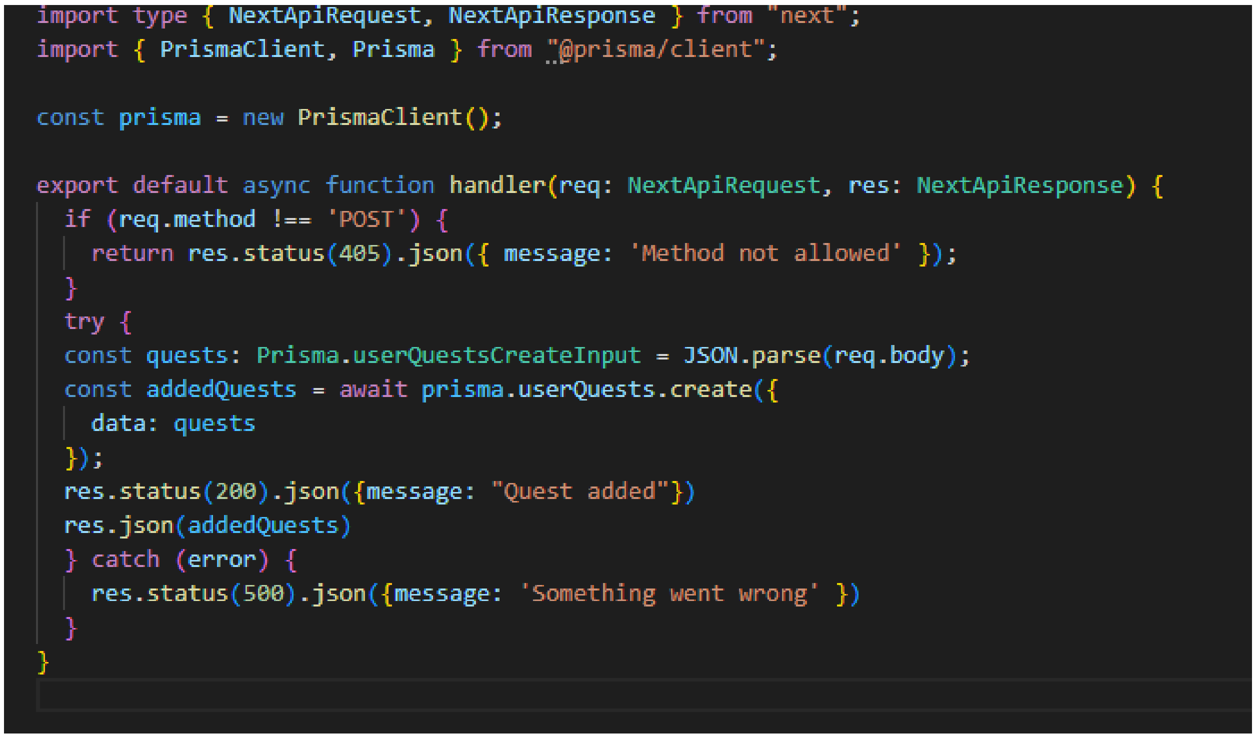
Create coins (Post)



```
import type { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  const body = req.body;
  try {
    const newCoin = await prisma.userCoins.create({
      data: {
        userCoinTotal: body.userCoinTotal,
        userID: body.userID
      }
    });
    return res.status(200).json({message: "Coins added"})
  } catch (error) {
    console.error("Request Error", error);
    res.status(500).json({ error: "Error creating coins", success:false });
  }
}
```

The purpose of this is to allow users to gain coins to spend on rewards once they complete a task.

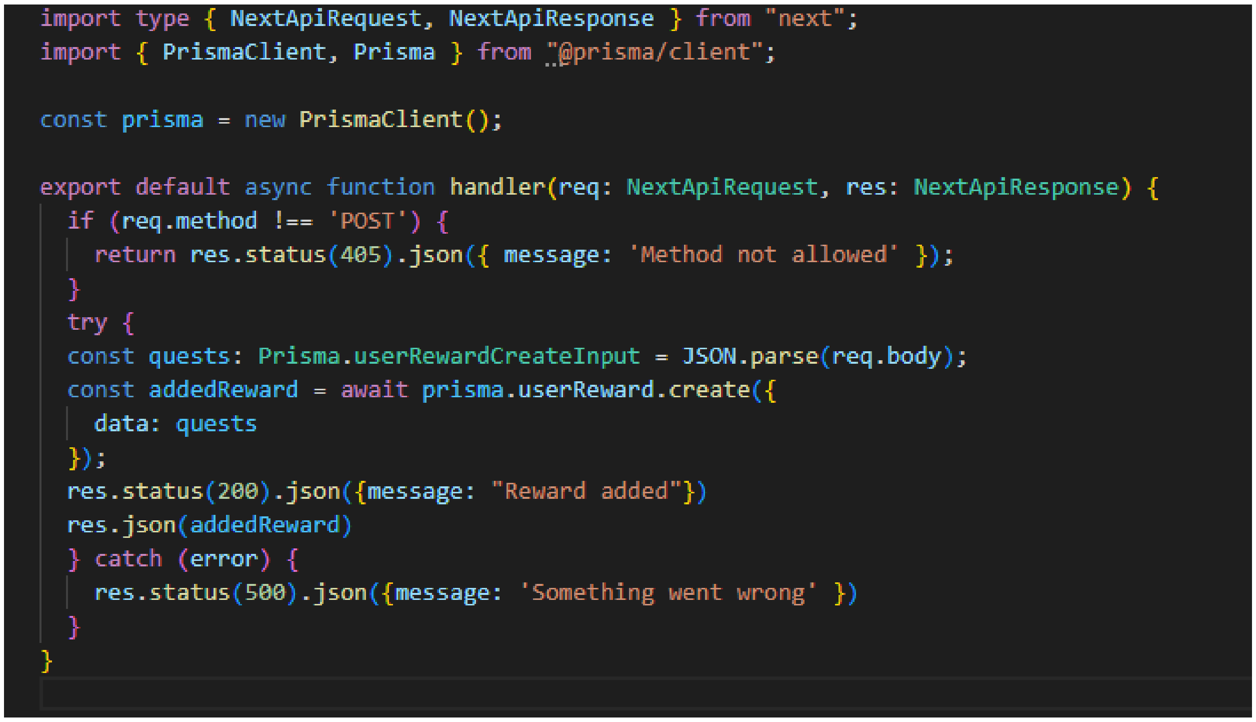
Create Quests (Post)



```
import type { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  try {
  const quests: Prisma.userQuestsCreateInput = JSON.parse(req.body);
  const addedQuests = await prisma.userQuests.create({
    data: quests
  });
  res.status(200).json({message: "Quest added"})
  res.json(addedQuests)
  } catch (error) {
    res.status(500).json({message: 'Something went wrong' })
  }
}
```

This allows the user to create their quests and the coin reward for each quest.

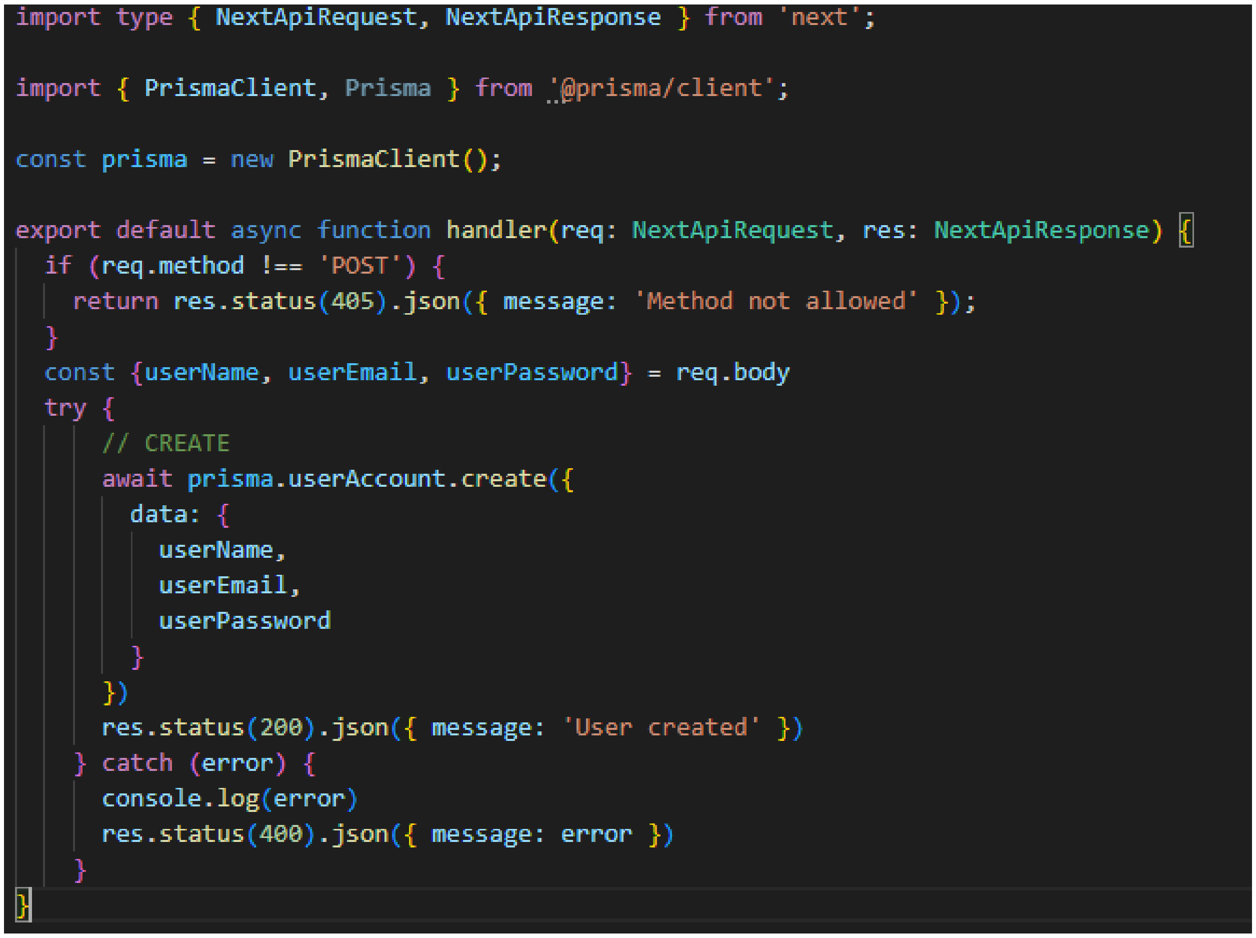
Create Rewards (Post)



```
import type { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  try {
  const quests: Prisma.userRewardCreateInput = JSON.parse(req.body);
  const addedReward = await prisma.userReward.create({
    data: quests
  });
  res.status(200).json({message: "Reward added"})
  res.json(addedReward)
  } catch (error) {
    res.status(500).json({message: 'Something went wrong' })
  }
}
```

The purpose is to allow the user to create rewards and their coin cost.

Create Account (Post)

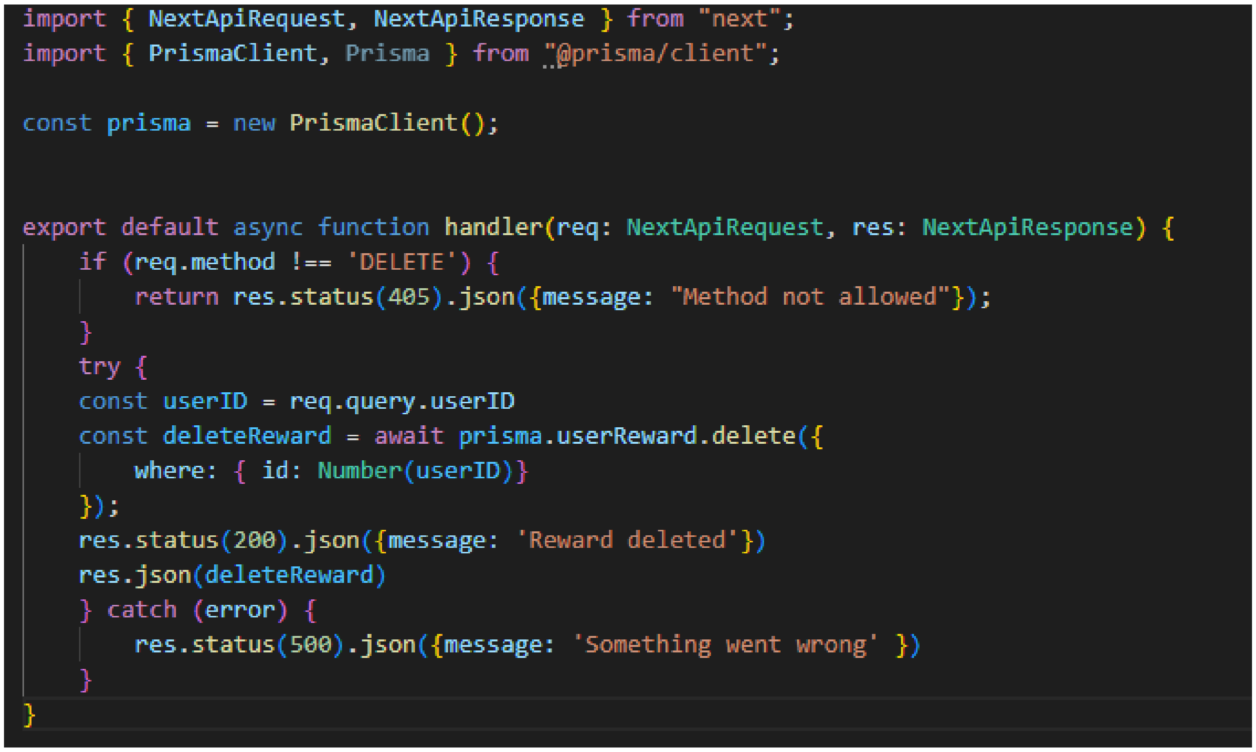


```
import type { NextApiRequest, NextApiResponse } from 'next';

import { PrismaClient, Prisma } from '@prisma/client';

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'POST') {
    return res.status(405).json({ message: 'Method not allowed' });
  }
  const {userName, userEmail, userPassword} = req.body
  try {
      // CREATE
      await prisma.userAccount.create({
        data: {
          userName,
          userEmail,
          userPassword
        }
      })
      res.status(200).json({ message: 'User created' })
    } catch (error) {
      console.log(error)
      res.status(400).json({ message: error })
    }
}
```

The purpose is to allow the user to create an account when logging on for the first time.

The Deletes

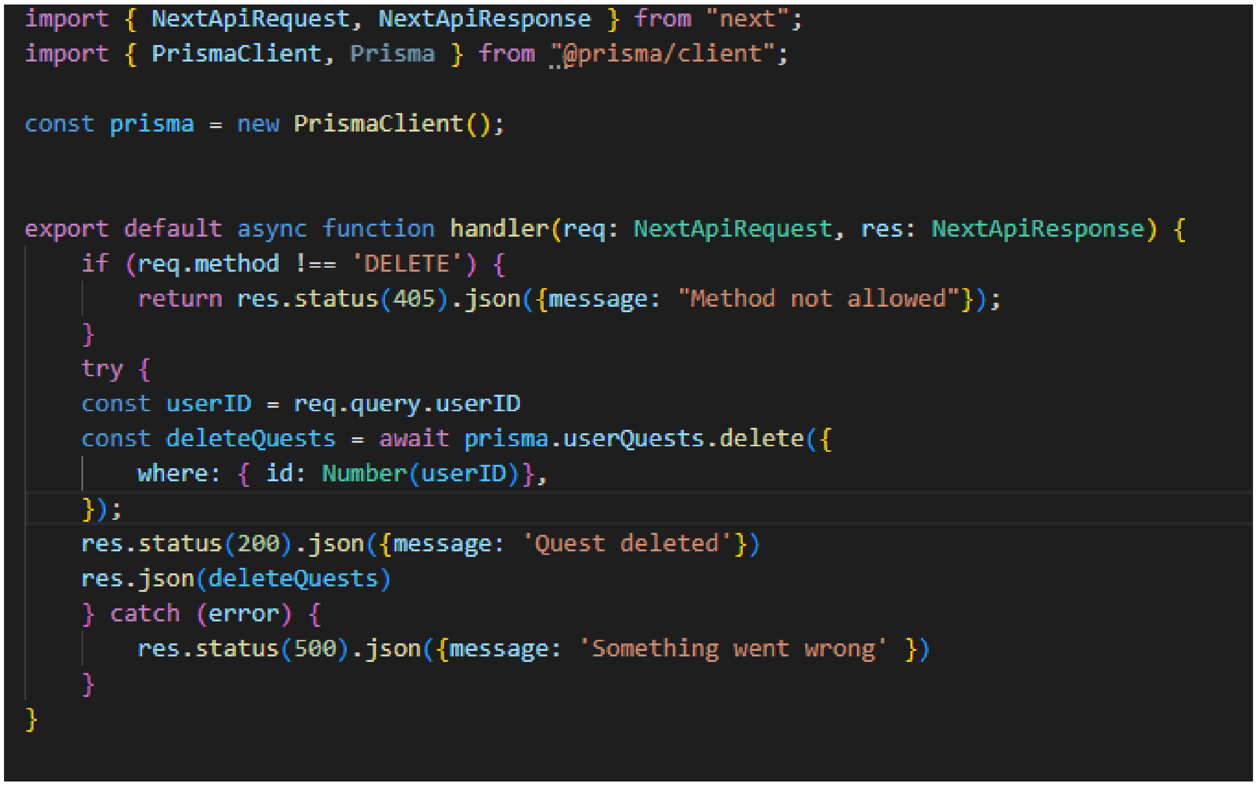
Delete Reward



```
import { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();


export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method !== 'DELETE') {
        return res.status(405).json({message: "Method not allowed"});
    }
    try {
    const userID = req.query.userID
    const deleteReward = await prisma.userReward.delete({
        where: { id: Number(userID)}
    });
    res.status(200).json({message: 'Reward deleted'})
    res.json(deleteReward)
    } catch (error) {
        res.status(500).json({message: 'Something went wrong' })
    }
}
```

The purpose is so that the user can delete rewards as they are completed or delete rewards they no longer wish to indulge in.

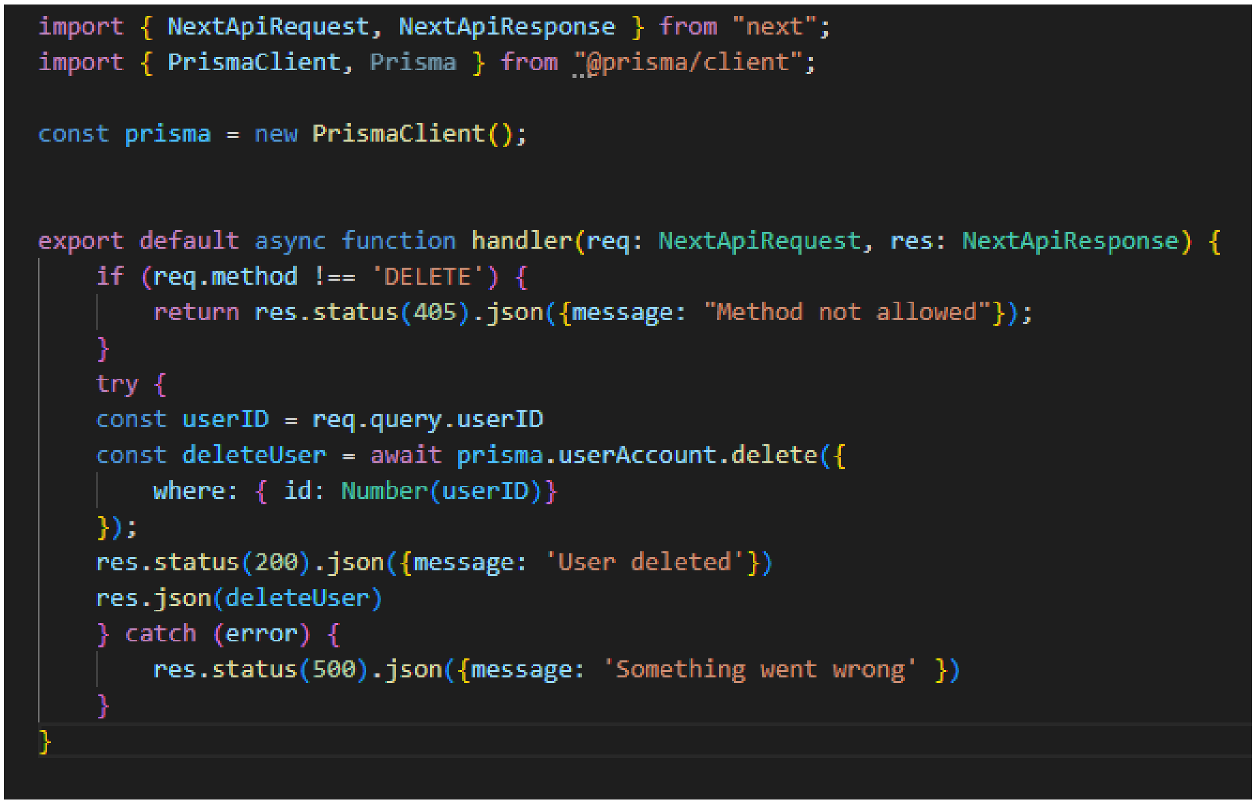
Delete Quests



```
import { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();


export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method !== 'DELETE') {
        return res.status(405).json({message: "Method not allowed"});
    }
    try {
    const userID = req.query.userID
    const deleteQuests = await prisma.userQuests.delete({
        where: { id: Number(userID)},
    });
    res.status(200).json({message: 'Quest deleted'})
    res.json(deleteQuests)
    } catch (error) {
        res.status(500).json({message: 'Something went wrong' })
    }
}
```

The purpose is to delete the quests as a user wishes or when the quest is complete.

Delete User Account



```
import { NextApiRequest, NextApiResponse } from "next";
import { PrismaClient, Prisma } from "@prisma/client";

const prisma = new PrismaClient();


export default async function handler(req: NextApiRequest, res: NextApiResponse) {
    if (req.method !== 'DELETE') {
        return res.status(405).json({message: "Method not allowed"});
    }
    try {
    const userID = req.query.userID
    const deleteUser = await prisma.userAccount.delete({
        where: { id: Number(userID)}
    });
    res.status(200).json({message: 'User deleted'})
    res.json(deleteUser)
    } catch (error) {
        res.status(500).json({message: 'Something went wrong' })
    }
}
```

The purpose is so that a user can delete their account if they no longer wish to use the application.

I know I didn’t have any responses in this first write up, however I will state my database is communicating with my application, it just isn’t saving the data in the correct way. I need to fix that tomorrow. I know you also mentioned I forgot to state how I am deploying the application, I will be using Vercel to deploy my application since I am using Nextjs.