File: /.env

Content:

```
MONGO_URI=

JWT_SECRET=
```

Directory: controllers

File: /controllers/coachController.js

Content:

```
const Coach = require('../models/Coach');

const { validationResult } = require('express-validator');

const asyncHandler = require('../middlewares/asyncHandler');

const Service = require('../models/Service');

// Get all coaches

exports.getAllCoaches = asyncHandler(async (req, res) => {

  const coaches = await Coach.find();

  res.json(coaches);

});


// Get a coach by ID

exports.getCoachById = asyncHandler(async (req, res) => {

  const coach = await Coach.findById(req.params.id);
  if (!coach) {
    return res.status(404).json({ msg: 'Coach not found' });
  }
  res.json(coach);
});

// Create a new coach
exports.createCoach = asyncHandler(async (req, res) => {
    try {
      const errors = validationResult(req);
      if (!errors.isEmpty()) {
        return res.status(400).json({ errors: errors.array() });
      }

         const { name, expertise, Service, bio, profilePicture, filters, socialMedia, faqs, username } =
req.body;

      console.log('Creating coach...');
      console.log('User ID:', req.user.id);
      console.log('User Type:', req.user.type);
      console.log('User Name:', req.user.username);

      const coach = new Coach({
        name,
        expertise,
        Service,
        bio,

        profilePicture,

        filters,

        socialMedia,

        faqs,

        username,

        user: req.user.id,

      });


      await coach.save();


      console.log('Coach created:', coach);


      res.status(201).json(coach);

    } catch (err) {

      console.error('Error creating coach:', err);

      res.status(500).json({ msg: 'Server error' });

    }

  });






// Update coach info

exports.updateCoach = asyncHandler(async (req, res) => {

  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const { name, expertise, bio, profilePicture, filters, socialMedia, faqs, username } = req.body;

  let coach = await Coach.findById(req.params.id);
  if (!coach) {
    return res.status(404).json({ msg: 'Coach not found' });
  }

  // Check if the authenticated user is the coach or an admin
  if (req.user.id !== coach.user.toString() && req.user.type !== 'Admin') {
    return res.status(401).json({ msg: 'Not authorized to update this coach' });
  }
console.log('Authenticated user ID:', req.user.id);
console.log('Coach user ID:', coach.user.toString());
console.log('Comparison result:', req.user.id !== coach.user.toString());


  coach.name = name;
  coach.expertise = expertise;
  coach.bio = bio;
  coach.profilePicture = profilePicture;
  coach.filters = filters;
  coach.socialMedia = socialMedia;
  coach.faqs = faqs;

  coach.username = username;

  // Only allow admins to update the gameId field

    if (req.user.type === 'Admin') {

        coach.gameId = gameId;

    }


  await coach.save();


  res.json(coach);

});
```

File: /controllers/gameController.js

Content:

```
const Game = require('../models/Game');


// Create a new game

exports.createGame = async (req, res) => {

  // Check for validation errors

  const errors = validationResult(req);

  if (!errors.isEmpty()) {

    return res.status(400).json({ errors: errors.array() });

  }
  try {
    // Create and save the game
    const game = new Game(req.body);
    await game.save();
    res.status(201).json(game);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Failed to create game' });
  }
};

// Get all games
exports.getAllGames = async (req, res) => {
  try {
    const games = await Game.find();
    res.json(games);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Failed to retrieve games' });
  }
};

// Get a game by ID
exports.getGameById = async (req, res) => {
  try {
    const game = await Game.findById(req.params.id);
    if (!game) {
      return res.status(404).json({ error: 'Game not found' });
    }
    res.json(game);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Failed to retrieve game' });
  }
};

// Update a game
exports.updateGame = async (req, res) => {
  // Check for validation errors
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  try {
    // Update the game
    const game = await Game.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
    });
    if (!game) {
      return res.status(404).json({ error: 'Game not found' });
    }
    res.json(game);

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Failed to update game' });

  }

};


// Delete a game

exports.deleteGame = async (req, res) => {

  try {

    const game = await Game.findByIdAndRemove(req.params.id);

    if (!game) {

      return res.status(404).json({ error: 'Game not found' });

    }

    res.json({ msg: 'Game deleted' });

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Failed to delete game' });

  }

};
```

File: /controllers/orderController.js

Content:

```
const asyncHandler = require('../middlewares/asyncHandler');
const Order = require('../models/Order');


// Function to calculate the commission
function calculateCommission(order) {
  if (!order.service || !order.service.price) {
    throw new Error('Invalid order structure. Missing service or price property.');
  }


  const commissionPercentage = 10;
  const commission = order.service.price * (commissionPercentage / 100);
  return commission;
}


// POST /orders
exports.createOrder = asyncHandler(async (req, res, next) => {
  const order = new Order(req.body);
  try {
    order.commission = calculateCommission(order);
  } catch (error) {
    return res.status(400).json({
      success: false,
      error: error.message
    });
  }


  await order.save();
  res.status(201).json({
    success: true,
    data: order
  });
});



// GET /orders
exports.getOrders = asyncHandler(async (req, res, next) => {
  const orders = await Order.find();

  res.status(200).json({
    success: true,
    data: orders
  });
});

// GET /orders/:id
exports.getOrder = asyncHandler(async (req, res, next) => {
  const order = await Order.findById(req.params.id);

  res.status(200).json({
    success: true,
    data: order
  });
```

```
});


// PUT /orders/:id

exports.updateOrder = asyncHandler(async (req, res, next) => {

  const order = await Order.findByIdAndUpdate(req.params.id, req.body, {

    new: true,

    runValidators: true

  });


  res.status(200).json({

    success: true,

    data: order

  });

});


// DELETE /orders/:id

exports.deleteOrder = asyncHandler(async (req, res, next) => {

  await Order.findByIdAndDelete(req.params.id);


  res.status(200).json({

    success: true,

    data: {}

  });

});
```

File: /controllers/serviceController.js

Content:

```
const asyncHandler = require('../middlewares/asyncHandler');
const Service = require('../models/Service');
const Coach = require('../models/Coach');
// Create a new service for a coach
exports.createService = asyncHandler(async (req, res) => {
  const { title, description, duration, price } = req.body;
  const coach = await Coach.findById(req.params.id);

  // Check if the authenticated user is the coach or an admin
  if (req.user.id !== coach.user.toString() && req.user.type !== 'Admin') {
    return res.status(401).json({ msg: 'Not authorized to create this service' });
  }

  const service = new Service({
    coach: coach._id,
    title,
    description,
    duration,
    price
  });

  await service.save();
  coach.services.push(service._id);
  await coach.save();
  res.status(201).json(service);
});

// Update a service for a coach
exports.updateService = asyncHandler(async (req, res) => {
  const { title, description, duration, price } = req.body;
  const service = await Service.findById(req.params.serviceId);
  const coach = await Coach.findById(service.coach);

  // Check if the authenticated user is the coach or an admin
  if (req.user.id !== coach.user.toString() && req.user.type !== 'Admin') {
    return res.status(401).json({ msg: 'Not authorized to update this service' });
  }

  service.title = title;
  service.description = description;
  service.duration = duration;
  service.price = price;
  await service.save();
  res.json(service);
});

// Delete a service for a coach
exports.deleteService = asyncHandler(async (req, res) => {
  const service = await Service.findById(req.params.serviceId);
  const coach = await Coach.findById(service.coach);
```

```
    // Check if the authenticated user is the coach or an admin
    if (req.user.id !== coach.user.toString() && req.user.type !== 'Admin') {
      return res.status(401).json({ msg: 'Not authorized to delete this service' });
    }

    await service.remove();
    coach.services.pull(service._id);
    await coach.save();
    res.status(204).json({ msg: 'Service deleted' });
  });
```

File: /controllers/userController.js

Content:

```
const User = require('../models/User');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { validationResult } = require('express-validator');

// Register a new user
exports.registerUser = async (req, res) => {
  const { username, email, password, type } = req.body;
```

```
try {
  // Check if the user already exists
  let user = await User.findOne({ email });

  if (user) {
    return res.status(400).json({ msg: 'User already exists' });
  }

  // Create a new user
  user = new User({
    username,
    email,
    password,
    type,
  });

  // Save the user to the database
  await user.save();

  // Create and return a JWT
  const payload = {
    user: {
      id: user.id,
    },
  };
```

```
    jwt.sign(
      payload,
      process.env.JWT_SECRET,
      { expiresIn: '1h' },
      (err, token) => {
        if (err) throw err;
        res.json({ token });
      }
    );
  } catch (err) {
    console.error(err.message);
    res.status(500).send('Server error');
  }
};

// Login a user
exports.loginUser = async (req, res) => {
  const { email, password } = req.body;

  try {
    // Check if the user exists
    let user = await User.findOne({ email }).select('+password');

    if (!user) {
      return res.status(400).json({ msg: 'Invalid Credentials' });
    }
    );

    // Check if the password is correct
    const isMatch = await bcrypt.compare(password, user.password);

    if (!isMatch) {
      return res.status(400).json({ msg: 'Invalid Credentials' });
    }

    // Return a JWT
    const payload = {
      user: {
        id: user.id,
      },
    };

    jwt.sign(
      payload,
      process.env.JWT_SECRET,
      { expiresIn: '1h' },
      (err, token) => {
        if (err) throw err;
        res.json({ token });
      }
    );
  } catch (err) {
    console.error(err.message);
    res.status(500).send('Server error');
  }
};

// Get a user's profile
exports.getUserProfile = async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select('-password');

    if (!user) {
      return res.status(404).json({ msg: 'User not found' });
    }

    res.json(user);
  } catch (err) {
    console.error(err.message);
    res.status(500).send('Server error');
  }
};

// Update a user's profile
exports.updateUserProfile = async (req, res) => {
  // Implementation here
};

// Delete a user
```

```
exports.deleteUser = async (req, res) => {
  // Implementation here
};
```

Directory: middlewares

File: /middlewares/asyncHandler.js

Content:

```
const asyncHandler = (fn) => (req, res, next) =>
  Promise.resolve(fn(req, res, next)).catch(next);

module.exports = asyncHandler;
```

File: /middlewares/auth.js

Content:

```
const jwt = require('jsonwebtoken');

module.exports = function (req, res, next) {
  // Get token from header
  const authHeader = req.header('Authorization');

  // Check if no token
```

```
if (!authHeader) {
  return res.status(401).json({ msg: 'No token, authorization denied' });
}


const token = authHeader.split(' ')[1]; // split "Bearer" from "Bearer {token}"


// Verify token
try {
  const decoded = jwt.verify(token, process.env.JWT_SECRET);


  console.log('Decoded token:', decoded);


  if (!decoded.user || !decoded.user.id) {
    console.log('Invalid user information in token');
    return res.status(401).json({ msg: 'Invalid user information in token' });
  }


  req.user = decoded.user;
  console.log('User object:', req.user);


  // Check if user is an admin
 // if (req.user.type !== 'Admin') {
 //   console.log('User is not authorized');
   // return res.status(403).json({ msg: 'User is not authorized' });
  //}
```

```
    console.log('Username:', req.user.username);

    console.log('User Type:', req.user.type);


    next();

  } catch (err) {

    console.error(err);

    res.status(401).json({ msg: 'Token is not valid' });

  }

};
```

File: /middlewares/idChecker.js

Content:

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


const validateGameId = async (req, res, next) => {

  try {

    const gameId = req.params.id;

    if (!mongoose.Types.ObjectId.isValid(gameId)) {

      return res.status(400).json({ error: 'Invalid game ID' });

    }

    next();

  } catch (err) {

    console.error(err);

    res.status(500).json({ error: 'Failed to validate game ID' });
  }
};

module.exports = validateGameId;
```

Directory: models

File: /models/Coach.js

Content:

```
const mongoose = require('mongoose');
const CoachSchema = new mongoose.Schema({
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  name: {
    type: String,
    required: [true, 'Please provide a name for the coach'],
  },
  expertise: {
    type: String,
    required: [true, 'Please provide the coach\'s area of expertise'],
  },
  bio: {
    type: String,
    required: [true, 'Please provide a bio for the coach'],
  },
  profilePicture: {
    type: String,
    required: [true, 'Please provide the URL of the coach\'s profile picture'],
  },
  filters: {
    type: [String],
    required: [true, 'Please provide the coach\'s filters'],
  },
  socialMedia: {
    type: {
      twitter: String,
      instagram: String,
      facebook: String,
    },
  },
  faqs: [
    {
      question: String,
      answer: String,
    },
  ],
  username: {
    type: String,

    required: [true, 'Please provide a username for the coach\'s URL'],

    unique: true,

  },

  gameId: {

    type: String,

    required: false,

  },

  services: [{

    type: mongoose.Schema.Types.ObjectId,

    ref: 'Service'

  }],

});


const Coach = mongoose.model('Coach', CoachSchema);


module.exports = Coach;
```

File: /models/Game.js

Content:

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


const GameFilterSchema = new mongoose.Schema({

  filterName: {
    type: String,
    required: [true, 'Please provide a name for the filter'],
  },
  filterOptions: {
    type: [String],
    required: [true, 'Please provide options for the filter'],
  },
});

const GameSchema = new mongoose.Schema({
  gameId: {
    type: String,
    required: [true, 'Please provide a game id'],
    unique: true,
  },
  gameName: {
    type: String,
    required: [true, 'Please provide a name for the game'],
  },
  gameTitle: {
    type: String,
    required: [true, 'Please provide a title for the game'],
  },
  gameDescription: {
    type: String,
    required: [true, 'Please provide a description for the game'],
  },
  friendlyUrl: {
    type: String,
    required: [true, 'Please provide a friendly URL for the game'],
    unique: true,
  },
  filters: [GameFilterSchema],
});

const Game = mongoose.model('Game', GameSchema);

module.exports = Game;
```

File: /models/Order.js

Content:

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

const OrderSchema = new mongoose.Schema({
  service: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Service',
    required: true
  },
  customer: {
```

```
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
},
coach: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'User',
  required: true
},
status: {
  type: String,
  enum: ['pending', 'paid', 'completed', 'cancelled'],
  default: 'pending'
},
createdAt: {
  type: Date,
  default: Date.now
},
updatedAt: {
  type: Date,
  default: Date.now
},
commission: {
  type: Number,
  required: true
}
```

```
});

module.exports = mongoose.model('Order', OrderSchema);
```

File: /models/Service.js

Content:

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

const ServiceSchema = new mongoose.Schema({
  coach: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Coach',
    required: true,
  },
  title: {
    type: String,
    required: [true, 'Please provide a title for the service'],
  },
  description: {
    type: String,
    required: [true, 'Please provide a description for the service'],
  },
  duration: {
    type: Number,
    required: [true, 'Please provide the duration for the service in minutes'],
  },
  price: {
    type: Number,
    required: [true, 'Please provide the price for the service'],
  },
});

const Service = mongoose.model('Service', ServiceSchema);

module.exports = Service;
```

File: /models/User.js

Content:

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  username: {
    type: String,
    required: [true, 'Please provide a username'],
    unique: true,
  },
  email: {
    type: String,

    required: [true, 'Please provide an email'],

    unique: true,

    match: [

      /^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]{2,7}$/,

      'Please provide a valid email',

    ],

  },

  password: {

    type: String,

    required: [true, 'Please provide a password'],

    minlength: 6,

    select: false,

  },

  type: {

    type: String,

    enum: ['Customer', 'Coach', 'Admin'],

    default: 'Customer',

  },

  resetPasswordToken: String,

  resetPasswordExpire: Date,

});


UserSchema.pre('save', async function (next) {

  if (!this.isModified('password')) {

    next();
  }

  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

UserSchema.methods.matchPasswords = async function (password) {
  return await bcrypt.compare(password, this.password);
};

const User = mongoose.model('User', UserSchema);

module.exports = User;
```

Directory: routes

File: /routes/coachRoutes.js

Content:

```
const express = require('express');
const { body } = require('express-validator');
const coachController = require('../controllers/coachController');
const auth = require('../middlewares/auth');
const asyncHandler = require('../middlewares/asyncHandler');
const router = express.Router();


// Get all coaches

router.get('/', coachController.getAllCoaches);


// Get a coach by ID

router.get('/:id', coachController.getCoachById);


// Create a new coach

router.post(

  '/',

  [

    auth,

    body('name').notEmpty().withMessage('Name is required'),

    body('expertise').notEmpty().withMessage('Expertise is required'),

    // Add more validation rules as needed

  ],

  coachController.createCoach

);


// Update coach info

router.put(

  '/:id',

  [

    auth,
    body('name').notEmpty().withMessage('Name is required'),

    body('expertise').notEmpty().withMessage('Expertise is required'),

    // Add more validation rules as needed

    body('gameId').if(body('gameId').exists()).custom((value, { req }) => {

      if (req.user.type !== 'Admin') {

        throw new Error('Not authorized to update gameId');

      }

      return true;

    }),

  ],

  coachController.updateCoach

);


module.exports = router;
```

File: /routes/gameRoutes.js

Content:

```
const express = require('express');

const { body } = require('express-validator');

const gameController = require('../controllers/gameController');

const auth = require('../middlewares/auth');

const validateGameId = require('../middlewares/idChecker');


const router = express.Router();
```

```
// Create a new game
router.post(
  '/',
  auth,
  [
    body('gameId').notEmpty().withMessage('Game ID is required'),
    body('gameName').notEmpty().withMessage('Game name is required'),
    // Add more validation rules as needed
  ],
  gameController.createGame
);

// Get all games
router.get('/', gameController.getAllGames);

// Get a game by ID
router.get('/:id', validateGameId, gameController.getGameById);

// Update a game
router.put(
  '/:id',
  auth,
  [
    body('gameId').notEmpty().withMessage('Game ID is required'),
    body('gameName').notEmpty().withMessage('Game name is required'),
    // Add more validation rules as needed
  ],
  validateGameId,
  gameController.updateGame
);

// Delete a game
router.delete('/:id', auth, validateGameId, gameController.deleteGame);

module.exports = router;
```

File: /routes/orderRoutes.js

Content:

```
const express = require('express');
const {
  createOrder,
  getOrders,
  getOrder,
  updateOrder,
  deleteOrder
} = require('../controllers/orderController');

const router = express.Router();
```

```
router.post('/', createOrder);
router.get('/', getOrders);
router.get('/:id', getOrder);
router.put('/:id', updateOrder);
router.delete('/:id', deleteOrder);

module.exports = router;
```

File: /routes/serviceRoutes.js

Content:

```
const express = require('express');
const { body } = require('express-validator');
const serviceController = require('../controllers/serviceController');
const auth = require('../middlewares/auth');
const asyncHandler = require('../middlewares/asyncHandler');

const router = express.Router();

// Create a new service for a coach
router.post(
  '/:id/service',
  [
    auth,
    body('title').notEmpty().withMessage('Title is required'),
    body('description').notEmpty().withMessage('Description is required'),
    body('duration').notEmpty().withMessage('Duration is required'),
    body('price').notEmpty().withMessage('Price is required'),
  ],
  serviceController.createService
);

// Update a service for a coach
router.put(
  '/:id/service/:serviceId',
  [
    auth,
    body('title').notEmpty().withMessage('Title is required'),
    body('description').notEmpty().withMessage('Description is required'),
    body('duration').notEmpty().withMessage('Duration is required'),
    body('price').notEmpty().withMessage('Price is required'),
  ],
  serviceController.updateService
);

// Delete a service for a coach
router.delete('/:id/service/:serviceId', auth, serviceController.deleteService);

module.exports = router;
```

File: /routes/userRoutes.js

Content:

```
const express = require('express');
const { body } = require('express-validator');
const router = express.Router();
const userController = require('../controllers/userController');
const auth = require('../middlewares/auth');
const asyncHandler = require('../middlewares/asyncHandler');

// Register a new user
router.post(
  '/register',
  [
    body('username').notEmpty().withMessage('Username is required'),
    body('email').isEmail().withMessage('Please provide a valid email'),
    body('password')
      .isLength({ min: 6 })
      .withMessage('Password must be at least 6 characters long'),
    // Add more validation rules as needed
  ],
  asyncHandler(userController.registerUser)
);

// Login a user
router.post(
  '/login',
  [
    body('email').isEmail().withMessage('Please provide a valid email'),
    body('password').notEmpty().withMessage('Password is required'),
  ],
  asyncHandler(userController.loginUser)
);

// Get user profile
router.get('/profile', auth, asyncHandler(userController.getUserProfile));

// Update user profile
router.put(
  '/profile',
  auth,
  [
    body('username').notEmpty().withMessage('Username is required'),
    body('email').isEmail().withMessage('Please provide a valid email'),
    // Add more validation rules as needed
  ],
  asyncHandler(userController.updateUserProfile)
);

// Delete a user
router.delete('/user', auth, asyncHandler(userController.deleteUser));

module.exports = router;
```

File: /server.js

Content:

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const mongoose = require('mongoose');

const app = express();
const port = process.env.PORT || 3000;

function connectDB() {
  console.log('Connecting to MongoDB...');
  console.log('MongoDB URI:', process.env.MONGO_URI);

  mongoose
    .connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    })
    .then(() => {
      console.log('Connected to MongoDB');

      app.listen(port, () => {
        console.log(`Server is running on port ${port}`);
      });
    })
    .catch((err) => {
      console.error('Failed to connect to MongoDB:', err);
      process.exit(1);
    });

  process.on('SIGINT', () => {
    mongoose.connection.close(() => {
      console.log('MongoDB connection closed');
      process.exit(0);
    });
  });
}

app.use(express.json());
app.use(helmet());
app.use(cors());
app.use((req, res, next) => {
  console.log('Request URL:', req.url);
  console.log('Authorization Header:', req.header('Authorization'));
  next();
});

const userRoutes = require('./routes/userRoutes');
```

```
app.use('/api/users', userRoutes);


const coachRoutes = require('./routes/coachRoutes');

app.use('/api/coaches', coachRoutes);


const serviceRoutes = require('./routes/serviceRoutes');

app.use('/api/services', serviceRoutes);


const gameRoutes = require('./routes/gameRoutes');

app.use('/api/games', gameRoutes);


const orderRoutes = require('./routes/orderRoutes');

app.use('/orders', orderRoutes);



app.get('/', (req, res) => {

  res.send('Hello, EGA!');

});


connectDB();
```