Backend 2.7_CW Exercises

ex01: user signup API

challenge

Develop a POST route at `/signup` that allows users to create new accounts by providing their user details. Utilize the `signup` function to save the user's information. Handle errors gracefully.

1. Create a POST Route: Design a POST route at `/signup` to handle requests for user signups.
2. Signup User: In the route handler, use the `signup` function to create a new user account based on the provided user details.
3. Success Response: If the user account is successfully created, respond with the saved user details.
4. Error Handling: If an error occurs during the process, respond with an error message.

example

```
POST /signup

Request Body:
{
  "email": "user@example.com",
  "password": "secretpassword",
  "username": "exampleuser",
  "profilePicture": "<https://example.com/profile.jpg>"
}
```

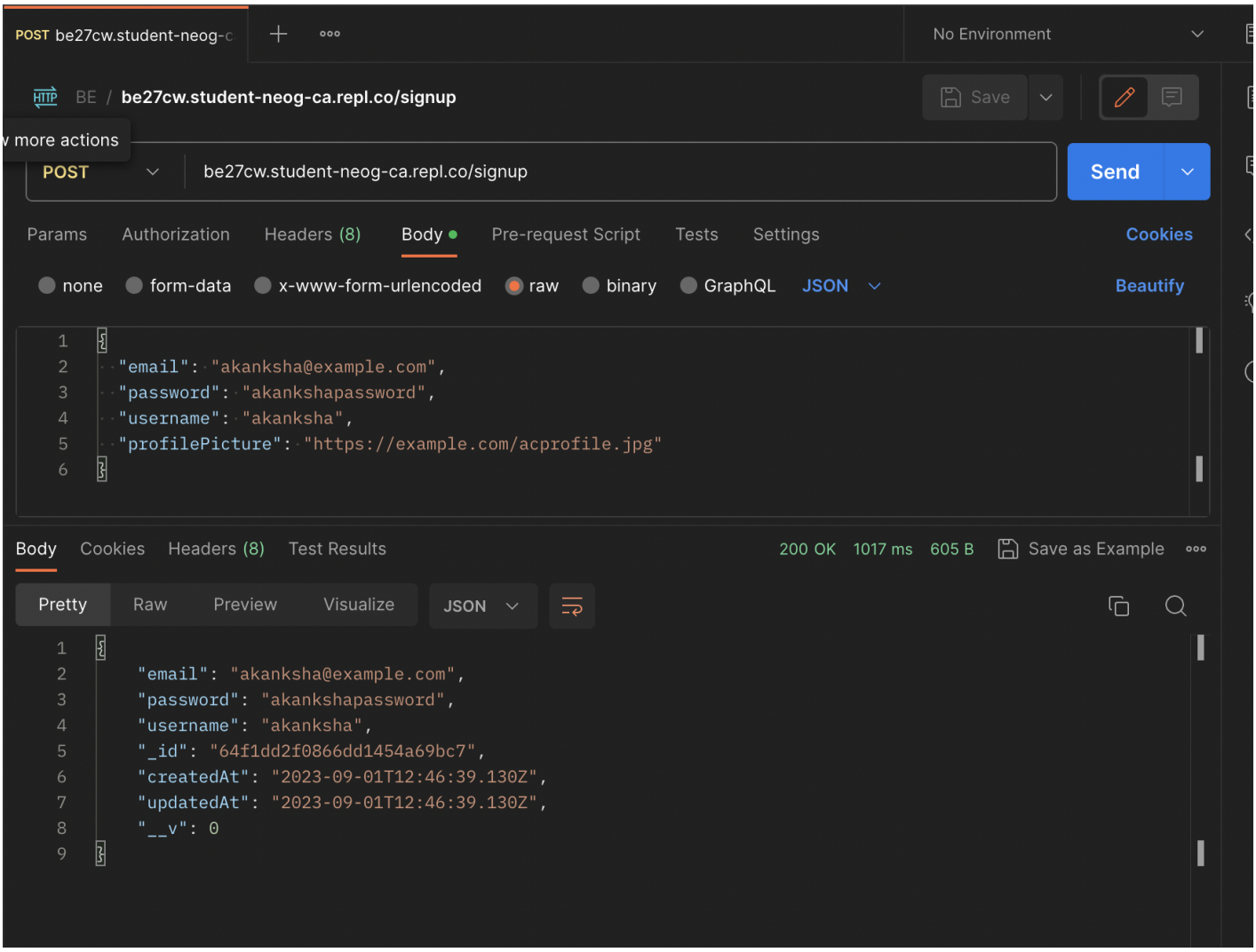
COPY

solution

https://replit.com/@tanaypratap/BE27CW-ex01

```
app.post('/signup', async (req, res) => {
  try {
    const savedUser = await signup(req.body)
    res.json(savedUser)
  } catch (error) {
    res.status(500).json({ error: 'Failed to create user account' })
  }
})
```

COPY



ex02: user login API

challenge

Develop a POST route at `/login` that allows users to log into their accounts by providing their email and password. Utilize the `login` function to authenticate users. Handle errors effectively.

1. Create a POST Route: Set up a POST route at `/login` to handle requests for user logins.
2. Login User: In the route handler, use the `login` function to authenticate the user based on the provided email and password.
3. Success Response: If the user is successfully authenticated, respond with the user's details.
4. Error Handling: If the credentials are invalid or an error arises, respond with an error message.

example

```
POST /login

Request Body:
{
```

```
  "email": "user@example.com",
  "password": "secretpassword"
}
```

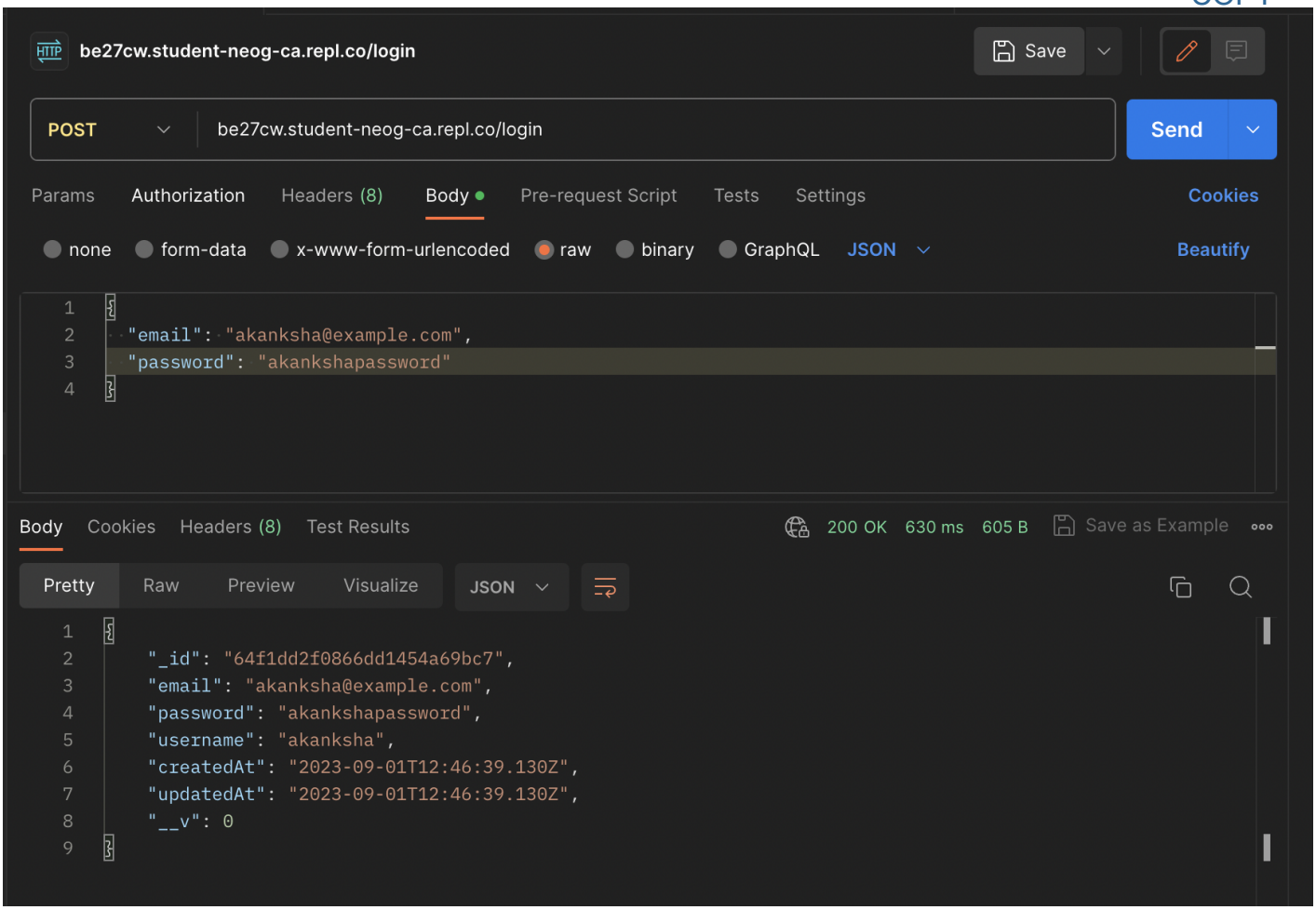
COPY

solution

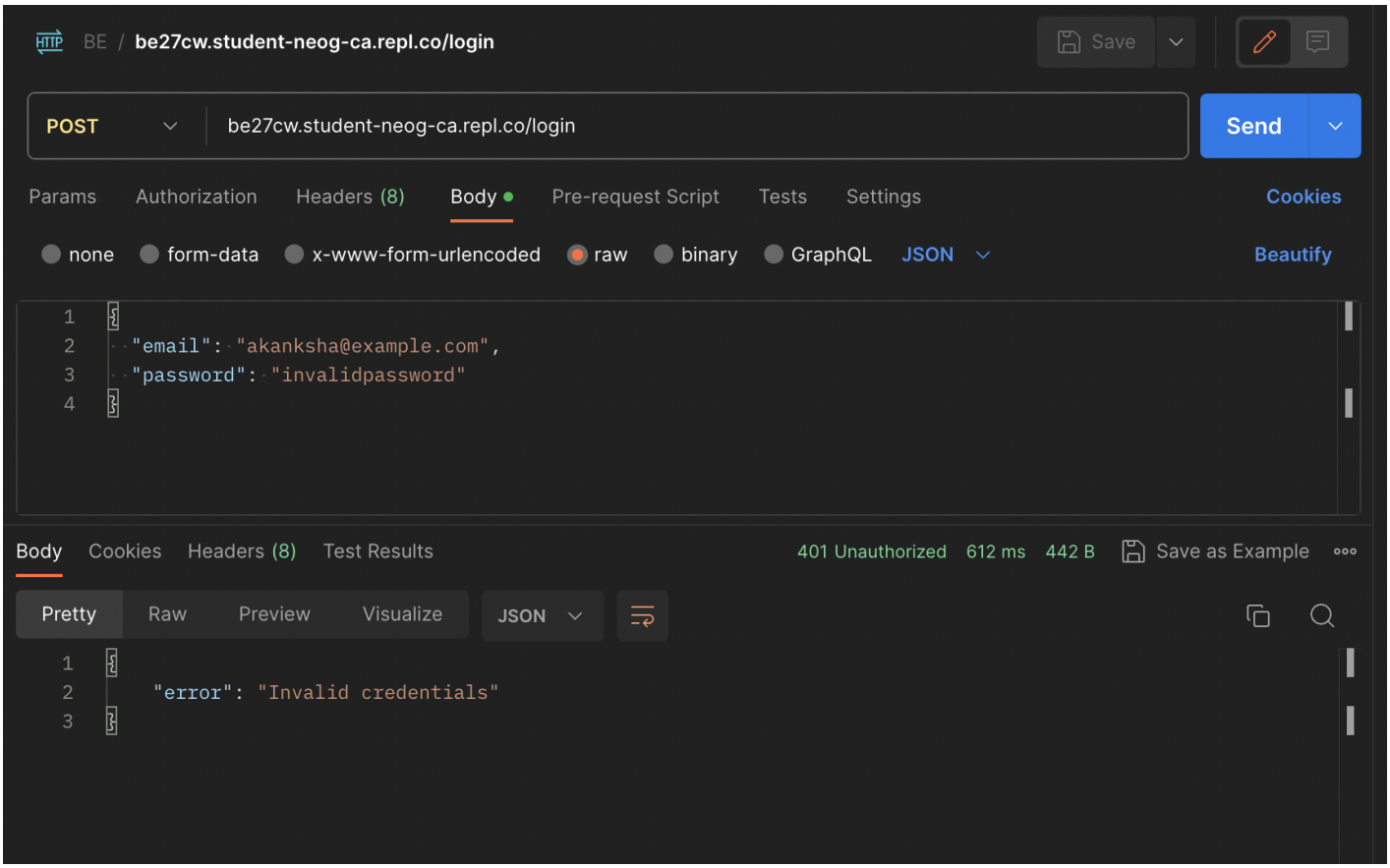
https://replit.com/@tanaypratap/BE27CW-ex02

```
app.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body
    const user = await login(email, password)
    res.json(user)
  } catch (error) {
    res.status(401).json({ error: 'Invalid credentials' })
  }
})
```

COPY



For Invalid credentials:



ex03: changing password API

challenge

Design a POST route at `/change-password` that enables users to update their passwords. Utilize the `changePassword` function to make the necessary changes. Handle errors proficiently.

1. Create a POST Route: Set up a POST route at `/change-password` to handle requests for changing user passwords.
2. Change Password: In the route handler, use the `changePassword` function to update the user's password if the current password is correct.
3. Success Response: If the password is successfully updated, respond with the updated user details.
4. Error Handling: If the credentials are invalid or an error arises, respond with an error message.

example

```
POST /change-password

Request Body:
{
  "email": "user@example.com",
  "currentPassword": "oldpassword",
  "newPassword": "newpassword"
```

```
}
```

COPY

solution

https://replit.com/@tanaypratap/BE27CW-ex03

```
app.post('/change-password', async (req, res) => {
  try {
    const { email, currentPassword, newPassword } = req.body
    const updatedUser = await changePassword(
      email,
      currentPassword,
      newPassword,
    )
    res.json(updatedUser)
  } catch (error) {
    res.status(401).json({ error: 'Invalid credentials' })
  }
})
```

COPY

ex04: updating Profile Picture API

challenge

Develop a POST route at `/update-profile-picture` that allows users to modify their profile pictures. Utilize the `updateProfilePicture` function to make the necessary changes. Handle errors effectively.

1. Create a POST Route: Design a POST route at `/update-profile-picture` to handle requests for updating user profile pictures.
2. Update Profile Picture: In the route handler, use the `updateProfilePicture` function to update the user's profile picture URL based on their email.
3. Success Response: If the profile picture is successfully updated, respond with the updated user details.
4. Error Handling: If the user is not found or an error arises, respond with an error message.

example

```
POST /update-profile-picture

Request Body:
{
  "email": "user@example.com",
  "newProfilePictureUrl": "<https://example.com/new-profile.jpg>"
}
```

COPY

solution

https://replit.com/@tanaypratap/BE27CW-ex04

```
app.post('/update-profile-picture', async (req, res) => {
  try {
    const { email, newProfilePictureUrl } = req.body
    const updatedUser = await updateProfilePicture(email, newProfilePictureUrl)
    res.json(updatedUser)
  } catch (error) {
    res.status(404).json({ error: 'User not found' })
  }
})
```

COPY

ex05: updating Contact Details API

challenge

Develop a POST route at `/update-contact/:email` that allows users to modify their contact details. Utilize the `updateContactDetails` function to make the necessary changes. Handle errors proficiently.

1. Create a POST Route: Design a POST route at `/update-contact/:email` to handle requests for updating user contact details.
2. Update Contact Details: In the route handler, use the `updateContactDetails` function to update the user's contact details based on their email.
3. Success Response: If the contact details are successfully updated, respond with the updated user details.
4. Error Handling: If the user is not found or an error arises, respond with an error message.

example

```
POST /update-contact/user@example.com

Request Body:
{
  "phoneNumber": "1234567890",
  "address": "123 Main St, City"
}

**Note: We are about to add new fields to the existing data, so make sure to add phoneNumber
```

COPY

solution

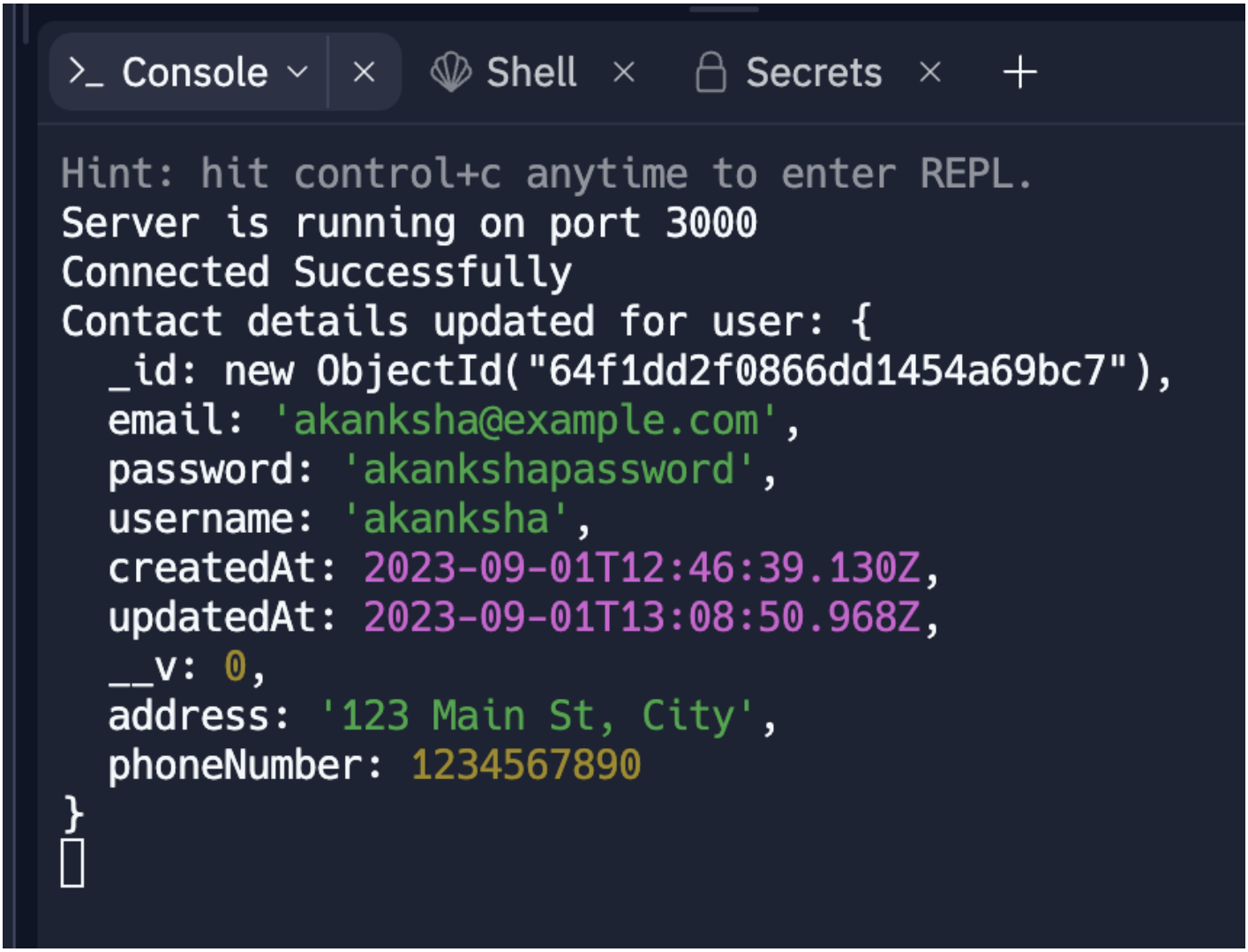
https://replit.com/@tanaypratap/BE27CW-ex05

```
async function updateContactDetails(email, updatedContactDetails) {
  try {
```

```
    const user = await User.findOne({ email })
    if (user) {
      Object.assign(user, updatedContactDetails)
      const updatedUser = await user.save()
      console.log('Contact details updated for user:', updatedUser)
    } else {
      throw new Error('User not found')
    }
  } catch (error) {
    throw error
  }
}

app.post('/update-contact/:email', async (req, res) => {
  try {
    const email = req.params.email
    const updatedContactDetails = req.body
    const updatedUser = await updateContactDetails(email, updatedContactDetails)
    res.json(updatedUser)
  } catch (error) {
    res.status(404).json({ error: 'User not found' })
  }
})
```

COPY



ex06: finding User by Phone Number API

challenge

Design a GET route at `/users/phone/:phoneNumber` that allows users to search for a user by their phone number. Utilize the `findUserByPhoneNumber` function to retrieve the user data. Handle errors effectively.

1. Create a GET Route: Set up a GET route at `/users/phone/:phoneNumber` to handle requests for finding a user by phone number.
2. Find User: In the route handler, use the `findUserByPhoneNumber` function to locate the user based on the provided phone number.
3. Success Response: If the user is found, respond with the user's details.
4. Error Handling: If the user is not found or an error arises, respond with an error message.

example

```
GET /users/phone/1234567890
```

COPY

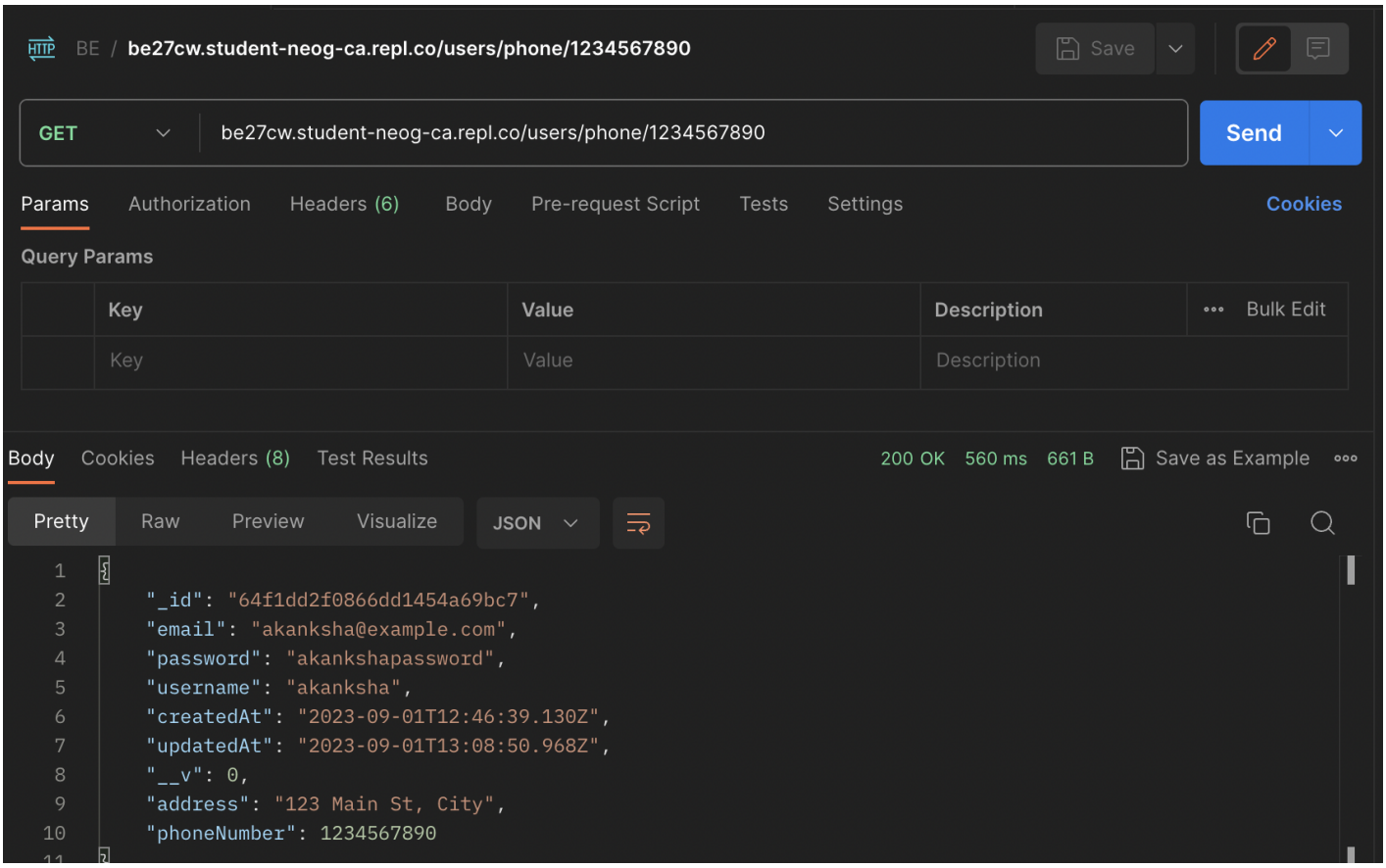
solution

https://replit.com/@tanaypratap/BE27CW-ex06

```
async function findUserByPhoneNumber(phoneNumber) {
  try {
    const userByPhoneNumber = await User.findOne({ phoneNumber })
    if (userByPhoneNumber) {
      return userByPhoneNumber
    } else {
      throw new Error('User not found')
    }
  } catch (error) {
    throw error
  }
}

app.get('/users/phone/:phoneNumber', async (req, res) => {
  try {
    const phoneNumber = req.params.phoneNumber
    const user = await findUserByPhoneNumber(Number(phoneNumber))
    if (user) {
      res.json(user)
    } else {
      res.status(404).json({ error: 'User not found' })
    }
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch user' })
  }
})
```

COPY



ex07: adding Rating and Review API

challenge

Develop a POST route at `/movies/:movieId/rating` that enables users to submit ratings and reviews for a movie. Utilize the `addRatingAndReview` function to handle the process. Handle errors effectively.

1. Create a POST Route: Set up a POST route at `/movies/:movieId/rating` to handle requests for adding ratings and reviews to a movie.
2. Add Rating and Review: In the route handler, use the `addRatingAndReview` function to add the provided rating and review to the specified movie.
3. Success Response: If the rating and review are successfully added, respond with the updated movie details including the populated reviews.
4. Error Handling: If the movie is not found or an error arises, respond with an error message.

example

```
POST /movies/61524b6b6a7a11abc1234567/rating

Request Body:
{
  "userId": "61524c806a7a11abc1234568", // use your created user's id
  "rating": 8,
```

```
        "review": "Good Movie"
}
```

COPY

solution

https://replit.com/@tanaypratap/BE27CW-ex07

```
app.post('/movies/:movieId/rating', async (req, res) => {
  try {
    const movieId = req.params.movieId
    const { userId, rating } = req.body

    const updatedMovie = await addRatingAndReview(
      movieId,
      userId,
      rating,
      review,
    )
    res.json(updatedMovie)
  } catch (error) {
    res.status(404).json({ error: 'Movie not found' })
  }
})
```

COPY

ex08: movie Reviews with User Details API

challenge

Develop a GET route at `/movies/:movieId/reviews` that allows users to fetch movie reviews with associated user details. Utilize the `getMovieReviewsWithUserDetails` function to handle the retrieval. Handle errors proficiently.

1. Create a GET Route: Set up a GET route at `/movies/:movieId/reviews` to handle requests for fetching movie reviews with user details.
2. Retrieve Reviews: In the route handler, use the `getMovieReviewsWithUserDetails` function to retrieve the movie reviews with associated user details for the specified movie.
3. Limit to Top 3 Reviews: Limit the reviews to the top 3, extracting the review text and user details.
4. Success Response: If the reviews are successfully fetched, respond with the array of reviews along with user details.
5. Error Handling: If the movie is not found or an error arises, respond with an error message.

example

```
GET /movies/61524b6b6a7a11abc1234567/reviews
```

COPY

solution

https://replit.com/@tanaypratap/BE27CW-ex08

```
app.get('/movies/:movieId/reviews', async (req, res) => {
  try {
    const movieId = req.params.movieId
    const reviewsWithUserDetails = await getMovieReviewsWithUserDetails(movieId)
    res.json(reviewsWithUserDetails)
  } catch (error) {
    res.status(404).json({ error: 'Movie not found' })
  }
})
```