One Login / Kutumbua

System Design and EGL Implementation

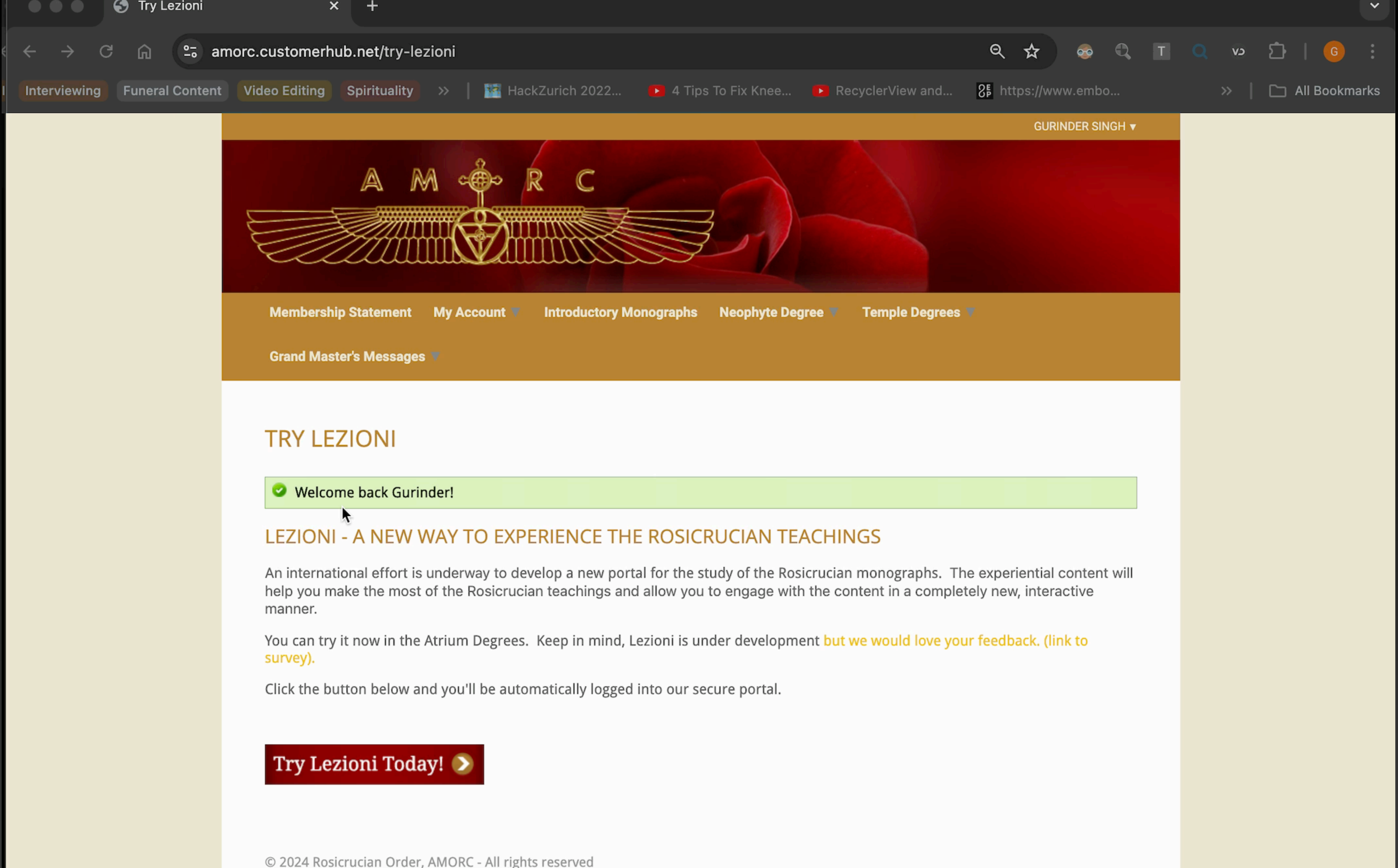
9-18-24

Introduction

- Objective: Unify authentication, member identification, and resource access worldwide.
- Current Challenge: Disconnected systems hindering seamless user experience.

Highways of connectivity

- Pre-Authorization
- Dynamic Authorization (API)
- Static Validation/Authorization
- New User



TRY LEZIONI

Welcome back Gurinder!

LEZIONI - A NEW WAY TO EXPERIENCE THE ROSICRUCIAN TEACHINGS

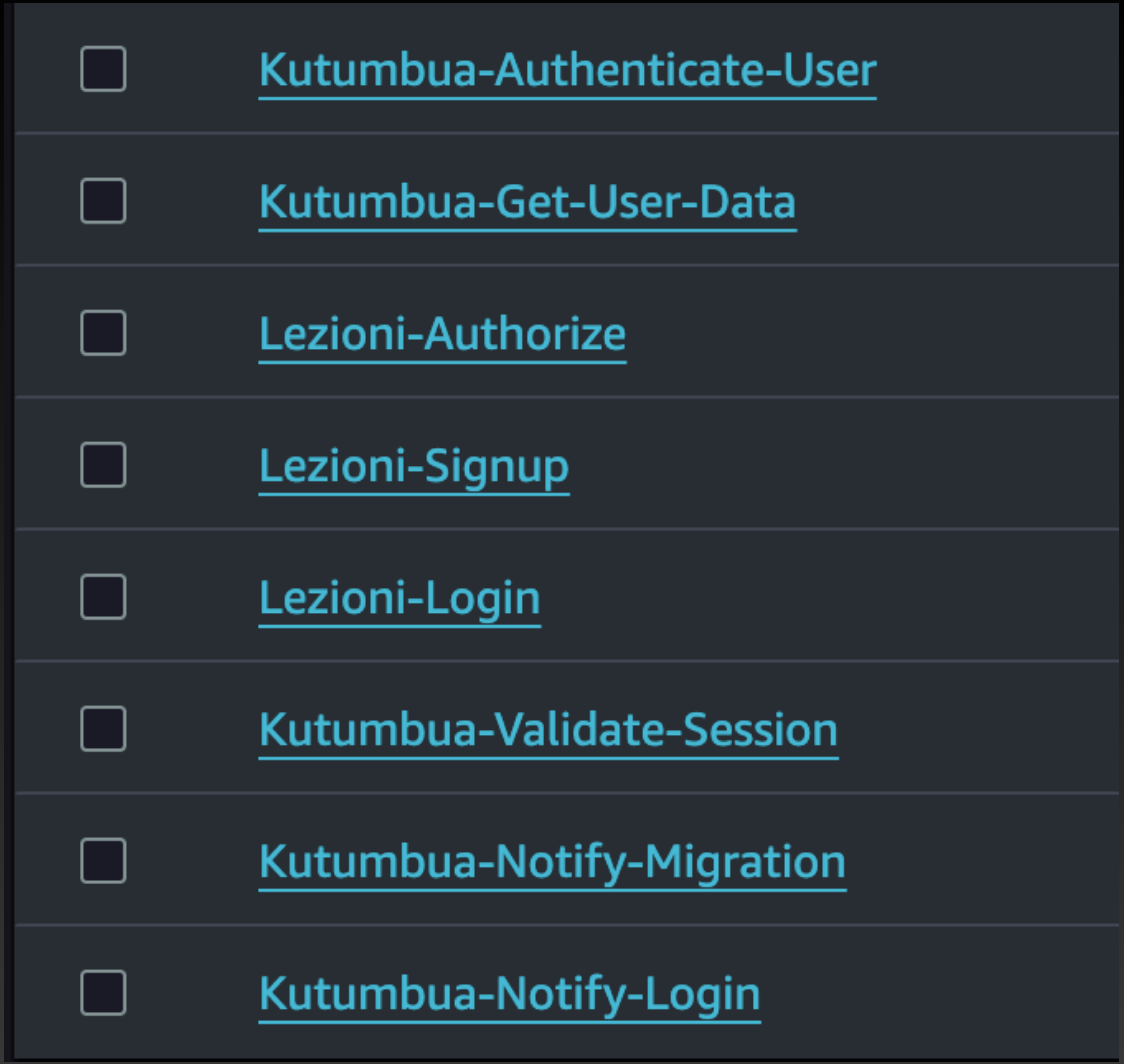
An international effort is underway to develop a new portal for the study of the Rosicrucian monographs. The experiential content will help you make the most of the Rosicrucian teachings and allow you to engage with the content in a completely new, interactive manner.

You can try it now in the Atrium Degrees. Keep in mind, Lezioni is under development but we would love your feedback. (link to survey).

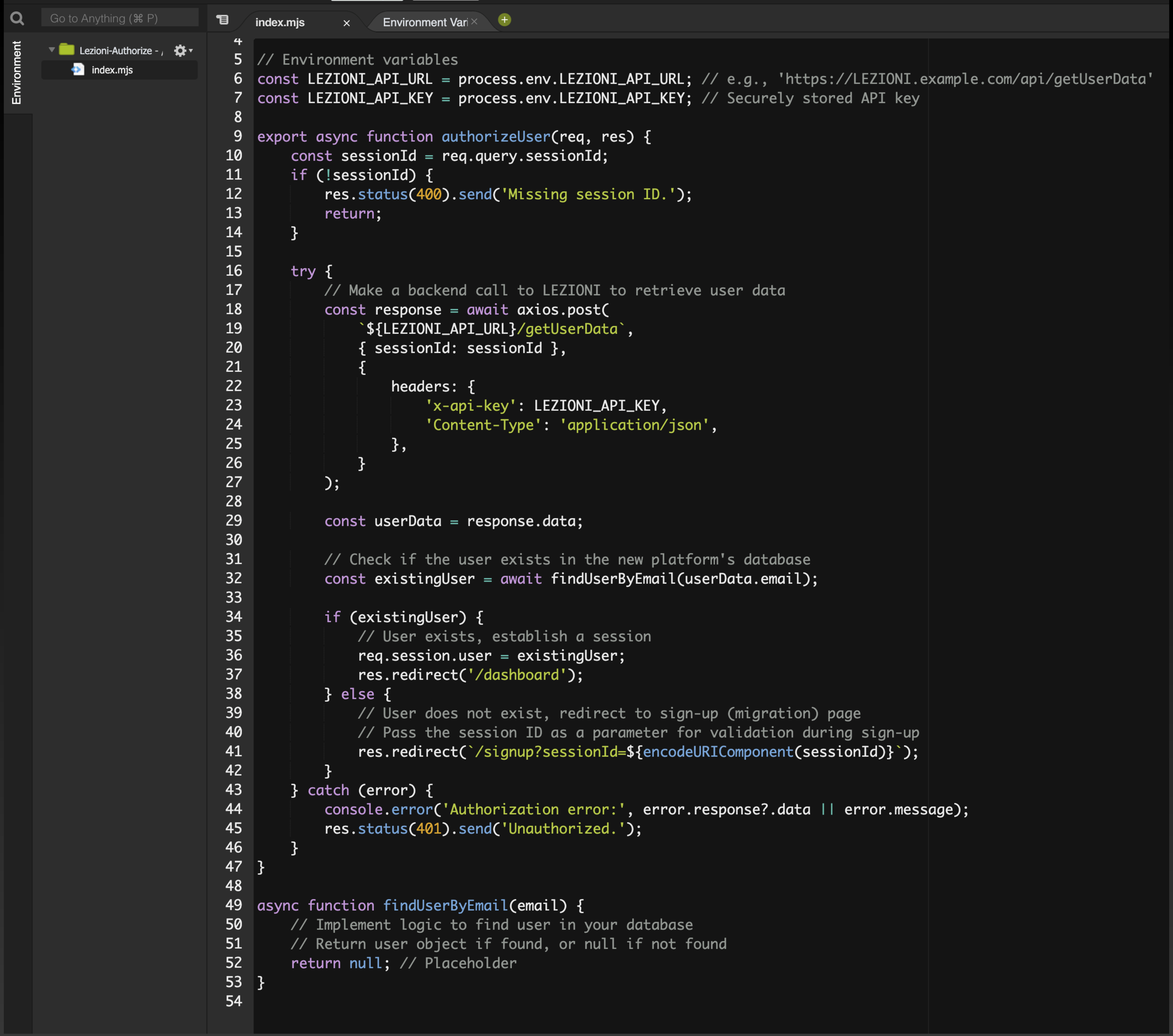
Click the button below and you'll be automatically logged into our secure portal.

Solution Overview

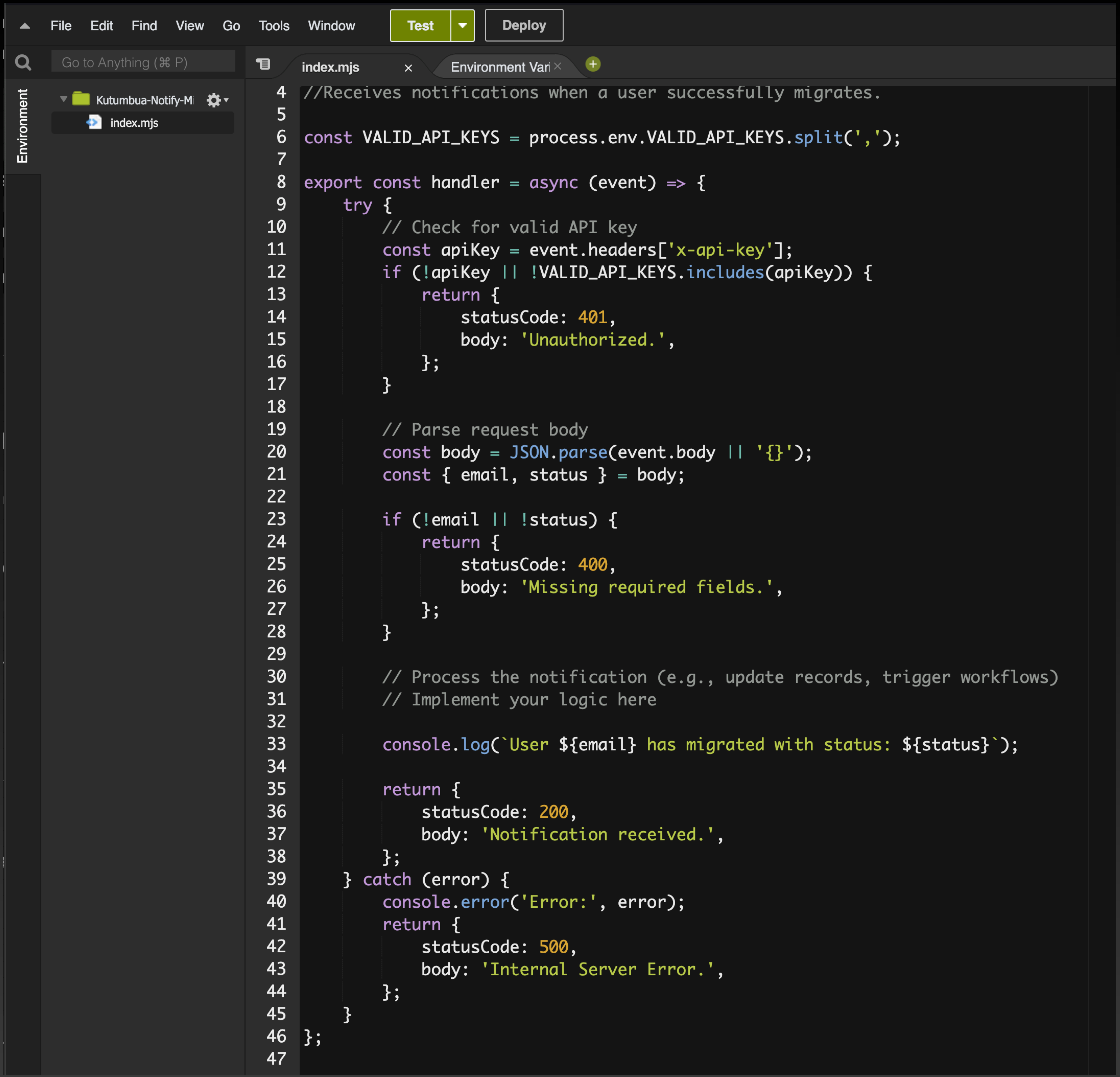
- Kutumbua: A centralized authentication and coordination service.
- Session ID Approach:
  - Secure authentication with backend validation using session IDs that abstract-away API/Bearer Tokens.
- Enhanced Integration:
  - Validation during sign-up and login.
  - Notification to Kutumbua upon successful migration or login.



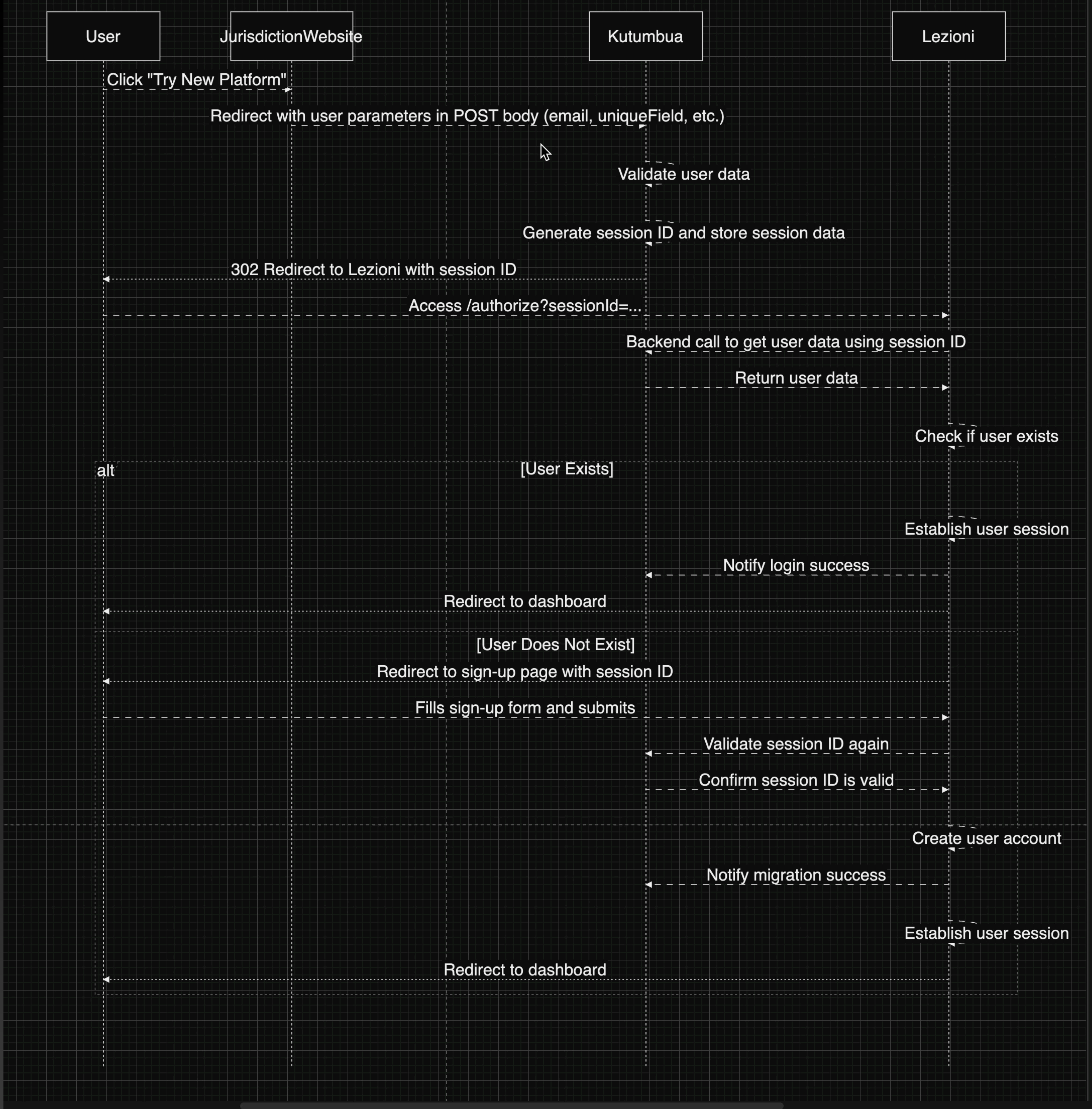
- [ ] Kutumbua-Authenticate-User
- [ ] Kutumbua-Get-User-Data
- [ ] Lezioni-Authorize
- [ ] Lezioni-Signup
- [ ] Lezioni-Login
- [ ] Kutumbua-Validate-Session
- [ ] Kutumbua-Notify-Migration
- [ ] Kutumbua-Notify-Login



```
// Environment variables
const LEZIONI_API_URL = process.env.LEZIONI_API_URL; // e.g., 'https://LEZIONI.example.com/api/getUserData'
const LEZIONI_API_KEY = process.env.LEZIONI_API_KEY; // Securely stored API key

export async function authorizeUser(req, res) {
    const sessionId = req.query.sessionId;
    if (!sessionId) {
        res.status(400).send('Missing session ID.');
        return;
    }

    try {
        // Make a backend call to LEZIONI to retrieve user data
        const response = await axios.post(
            `${LEZIONI_API_URL}/getUserData`,
            { sessionId: sessionId },
            {
                headers: {
                    'x-api-key': LEZIONI_API_KEY,
                    'Content-Type': 'application/json',
                },
            }
        );

        const userData = response.data;

        // Check if the user exists in the new platform's database
        const existingUser = await findUserByEmail(userData.email);

        if (existingUser) {
            // User exists, establish a session
            req.session.user = existingUser;
            res.redirect('/dashboard');
        } else {
            // User does not exist, redirect to sign-up (migration) page
            // Pass the session ID as a parameter for validation during sign-up
            res.redirect(`/signup?sessionId=${encodeURIComponent(sessionId)}`);
        }
    } catch (error) {
        console.error('Authorization error:', error.response?.data || error.message);
        res.status(401).send('Unauthorized.');
    }
}

async function findUserByEmail(email) {
    // Implement logic to find user in your database
    // Return user object if found, or null if not found
    return null; // Placeholder
}
```



```javascript
//Receives notifications when a user successfully migrates.

const VALID_API_KEYS = process.env.VALID_API_KEYS.split(',');

export const handler = async (event) => {
    try {
        // Check for valid API key
        const apiKey = event.headers['x-api-key'];
        if (!apiKey || !VALID_API_KEYS.includes(apiKey)) {
            return {
                statusCode: 401,
                body: 'Unauthorized.',
            };
        }

        // Parse request body
        const body = JSON.parse(event.body || '{}');
        const { email, status } = body;

        if (!email || !status) {
            return {
                statusCode: 400,
                body: 'Missing required fields.',
            };
        }

        // Process the notification (e.g., update records, trigger workflows)
        // Implement your logic here

        console.log(`User ${email} has migrated with status: ${status}`);

        return {
            statusCode: 200,
            body: 'Notification received.',
        };
    } catch (error) {
        console.error('Error:', error);
        return {
            statusCode: 500,
            body: 'Internal Server Error.',
        };
    }
};

```


User
JurisdictionWebsite
Kutumbua
Lezioni
Click "Try New Platform"
Redirect with user parameters in POST body (email, uniqueField, etc.)
Validate user data
Generate session ID and store session data
302 Redirect to Lezioni with session ID
Access /authorize?sessionId=...
Backend call to get user data using session ID
Return user data
Check if user exists
alt
[User Exists]
Establish user session
Notify login success
Redirect to dashboard
[User Does Not Exist]
Redirect to sign-up page with session ID
Fills sign-up form and submits
Validate session ID again
Confirm session ID is valid
Create user account
Notify migration success
Establish user session
Redirect to dashboard

High-Level Flow

1. User clicks "Try New Platform" on their local site.
2. Kutumbua validates the user and generates a session ID.
3. Kutumbua redirects the user to the new platform with the session ID.
4. The new platform retrieves user data from Kutumbua using the session ID.
5. The new platform checks if the user exists.
6. If user exists:
7. User is logged in.
8. New platform notifies Kutumbua of successful login.
9. If user does not exist:
10. User is directed to sign-up page.
11. Session ID is validated again during sign-up.
12. Upon successful migration, new platform notifies Kutumbua.
13. User lands on the dashboard, authenticated.

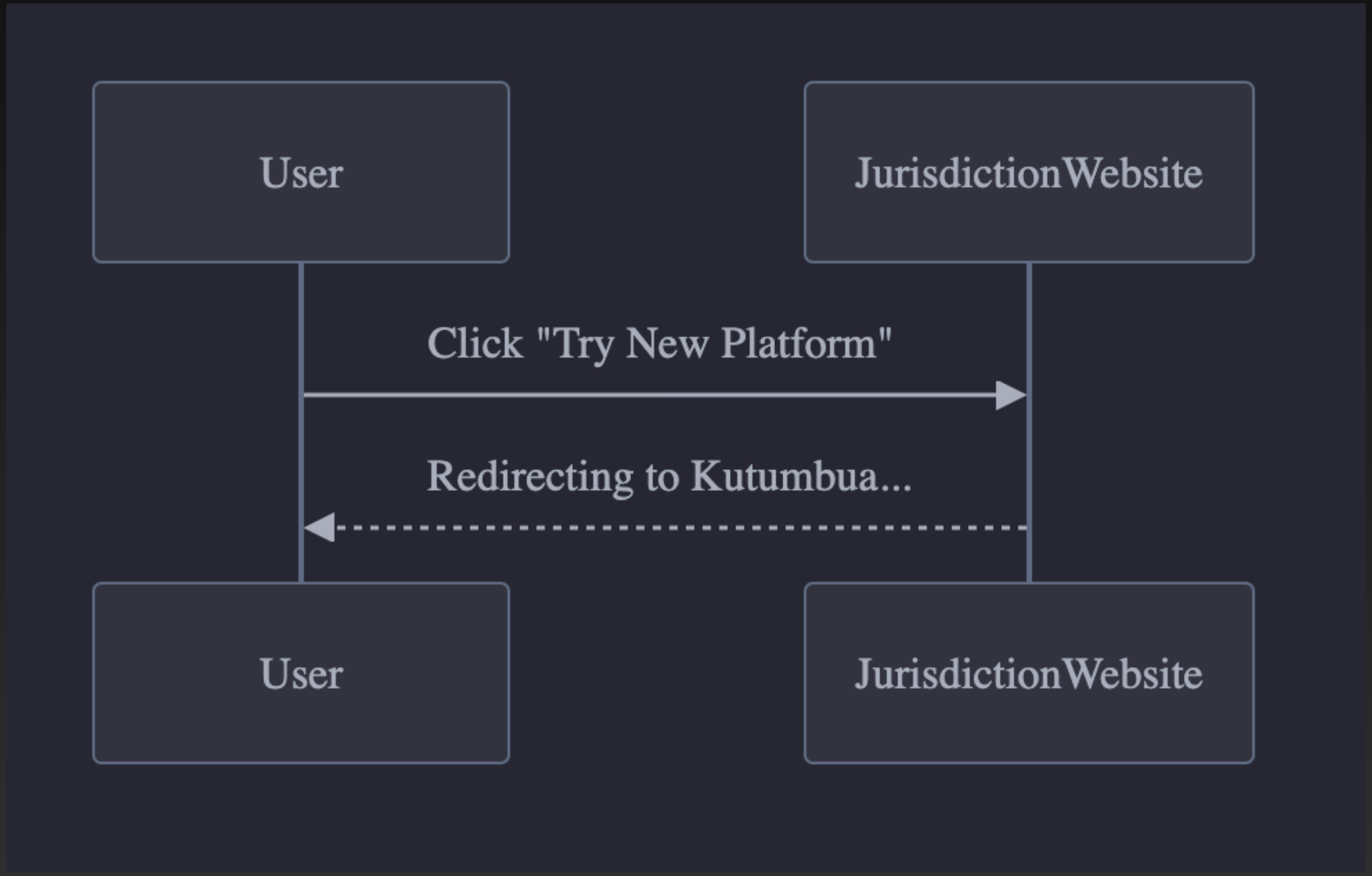
Why We Chose This Approach

- Security Enhancements:
  - Session ID validation at critical points.
  - Backend communication reduces exposure.
- Improved User Experience:
  - Seamless transitions. Users on Jurisdiction websites see a single redirect
  - Consistent authentication flow.
- Operational Benefits:
  - Real-time updates to Kutumbua.
  - Better tracking of user migration and login activities.

High-Level Flow

User Initiates Authentication via “Try Lezioni” button

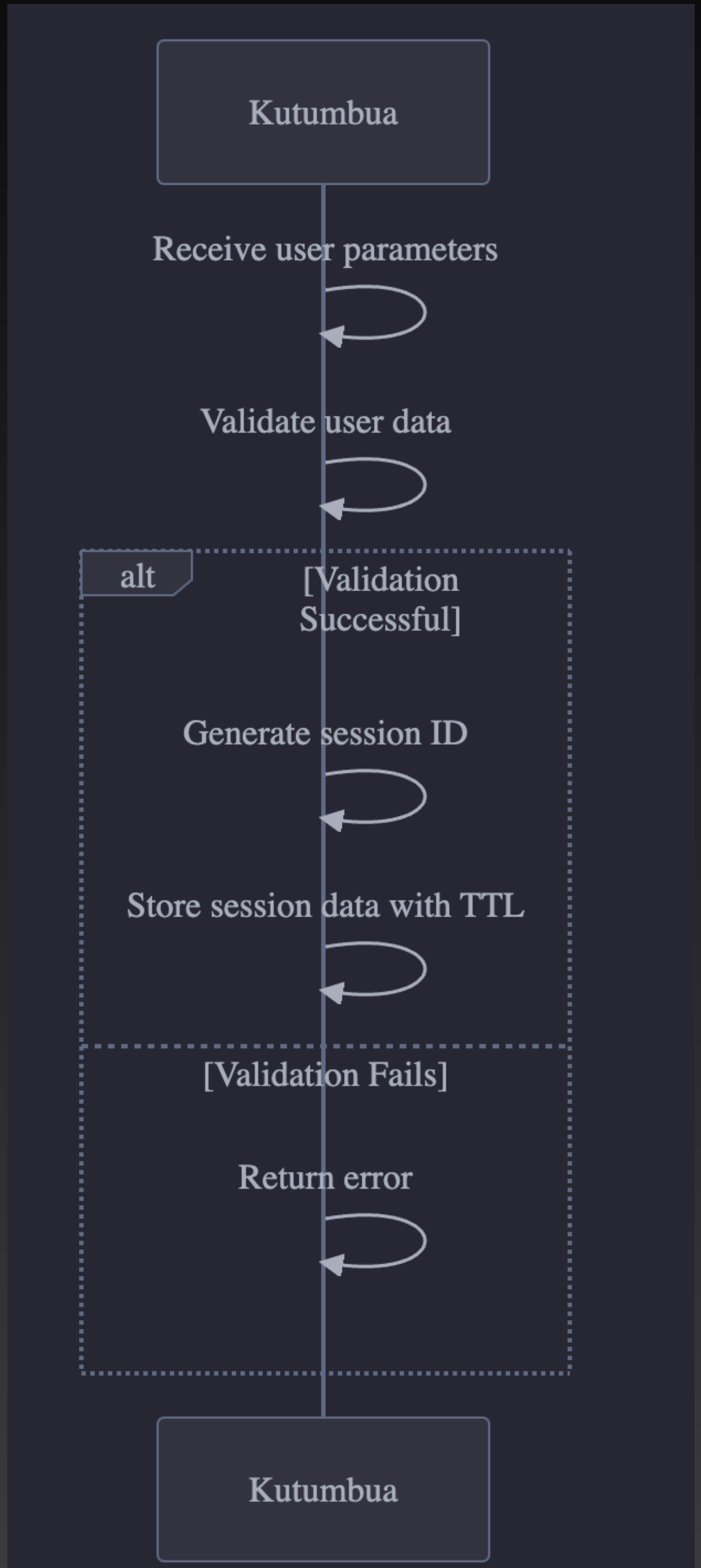
1. Action: User clicks the link.
2. Details: Minimal parameters passed.
3. Security: Sensitive data is protected.



High-Level Flow

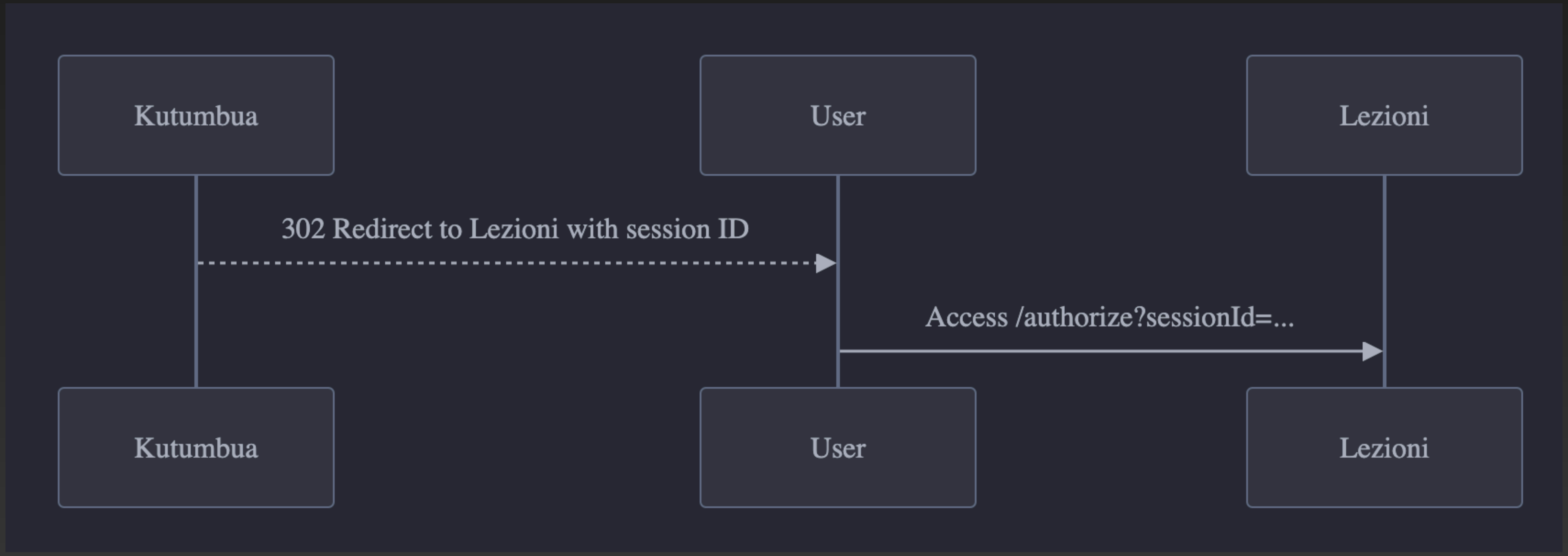
Kutumbua Validates and Generates Session ID

1. Action: Validation and session ID generation.
2. Security: Ensures only authorized users proceed.



High-Level Flow
Kutumbua Redirects User

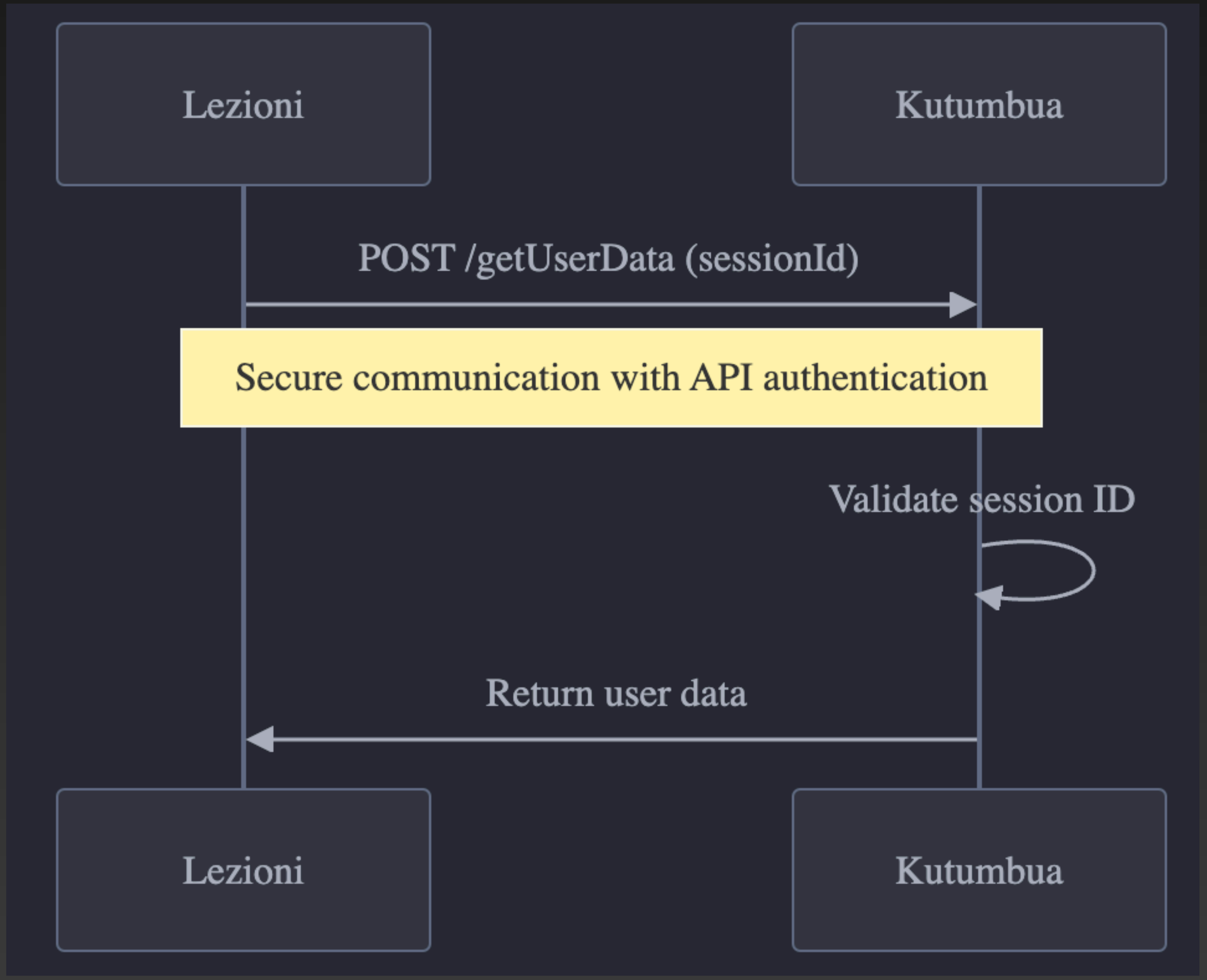
1. Action: 302 redirect with session ID.
2. User Experience: Smooth transition to the new platform.


Kutumbua
User
Lezioni
302 Redirect to Lezioni with session ID
Access /authorize?sessionId=...
Kutumbua
User
Lezioni

High-Level Flow

New Platform Retrieves User Data

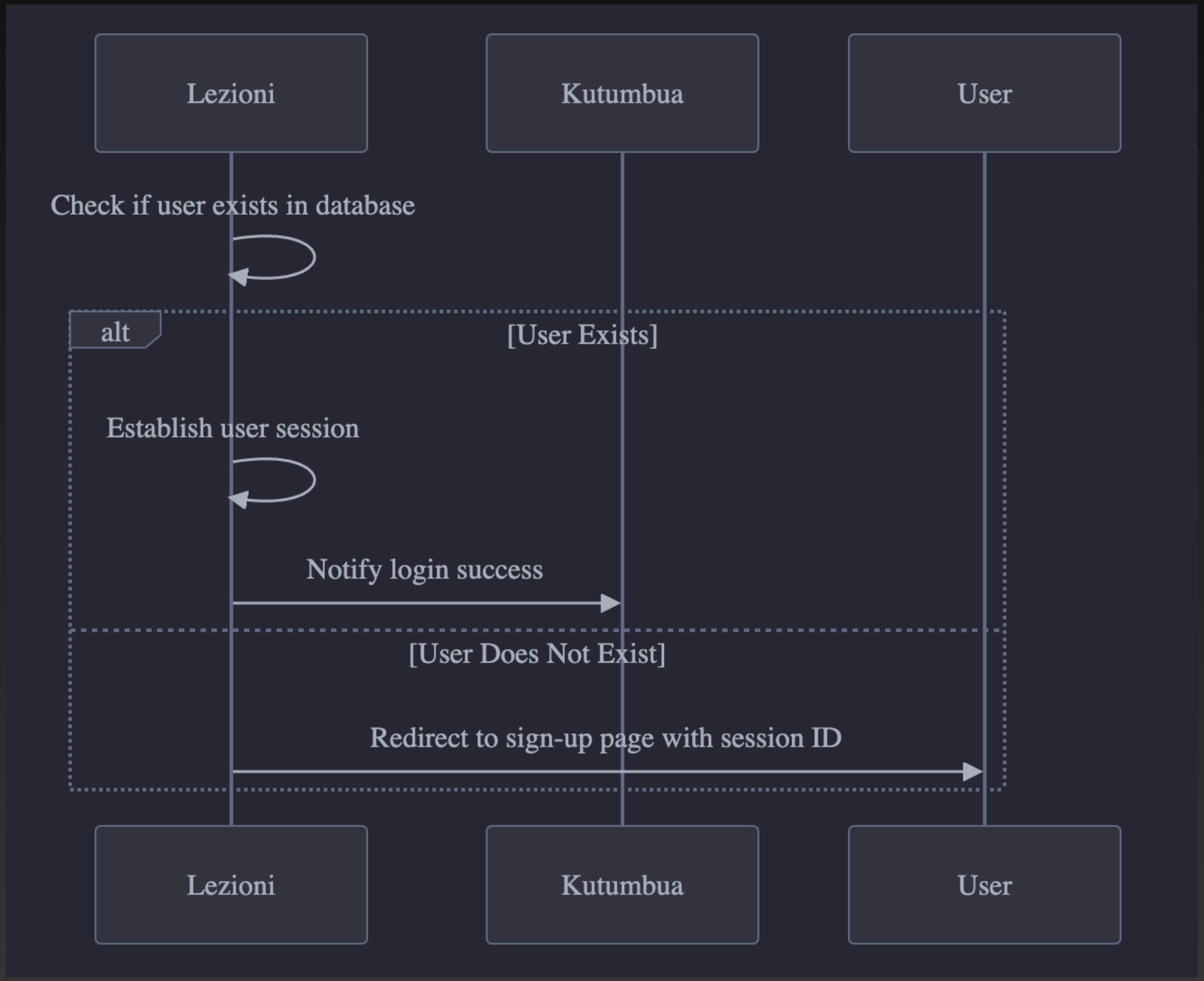
1. Action: Backend call to Kutumbua.
2. Security: Encrypted communication and API authentication.


Lezioni
Kutumbua
POST /getUserData (sessionId)
Secure communication with API authentication
Validate session ID
Return user data
Lezioni
Kutumbua

High-Level Flow

User Existence Check

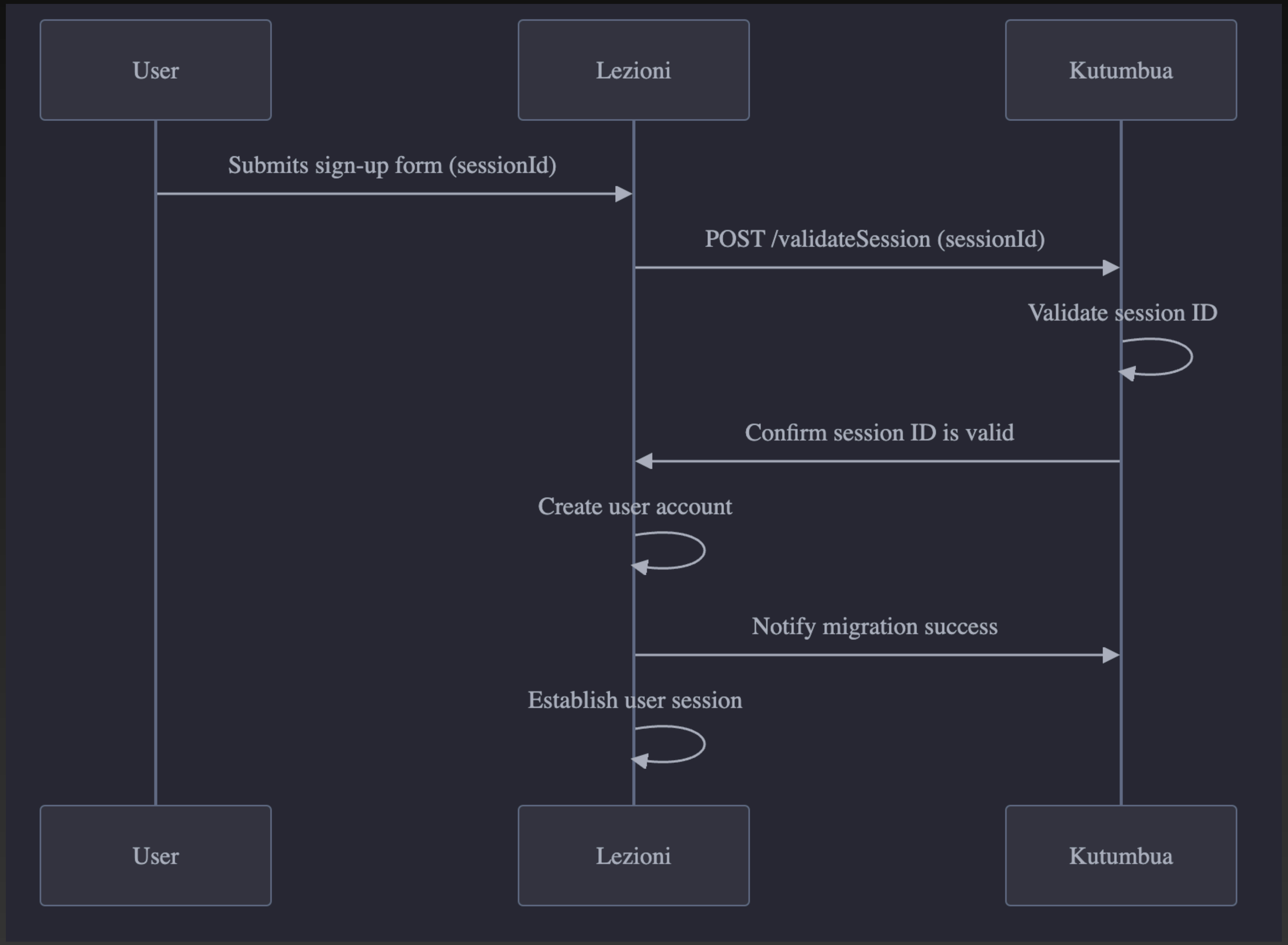
1. Action: Platform checks if user exists.
2. Paths:
   - User Exists: Proceed to login.
   - User Does Not Exist: Redirect to sign-up.



High-Level Flow

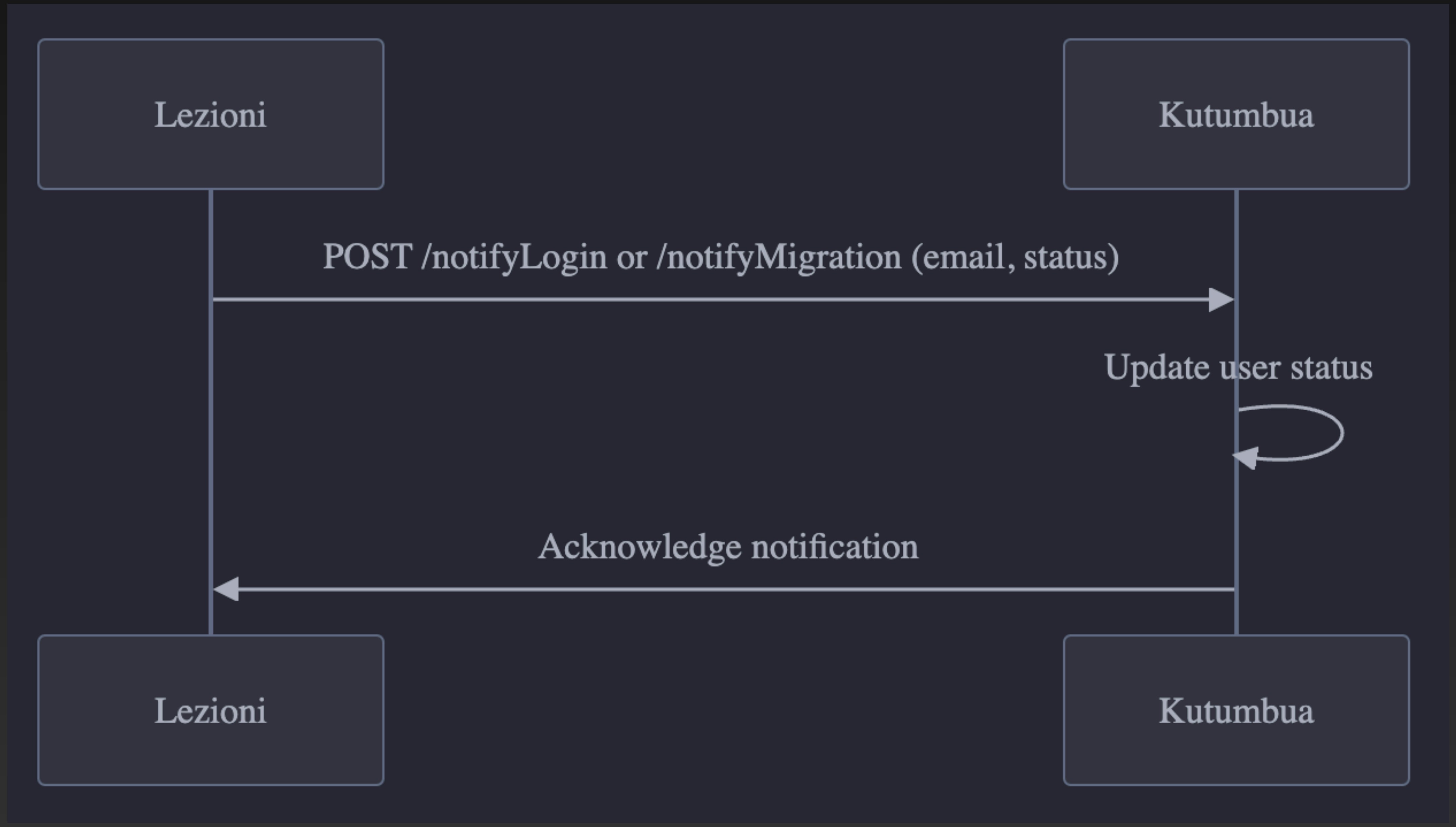
Sign-Up with Session ID Validation

1. Action: User fills sign-up form.
2. Security: Session ID re-validation with Kutumbua.
3. Integration: Upon success, platform notifies Kutumbua.



High-Level Flow
Successful Login or Migration Notification

1. Action: Platform notifies Kutumbua of user status.
2. Benefits:
   - Real-time updates: Kutumbua stays informed.
   - Enhanced tracking: Better user-management across systems.



Recap

1. Session ID Validation:
   - At sign-up and login.
   - Prevents replay attacks.
2. Backend Notifications:
   - Secured with API keys.
   - Confirm user status changes.
3. Data Protection:
   - Encrypted storage.
   - Secure API communications.

Security Measures

1. Monitoring Enhancements:
   - Track user migration and login patterns.
   - Identify and address bottlenecks.
2. Security Updates:
   - Regular API key/Certificate rotations.
   - Implement additional authentication factors.
3. Future Developments:
   - Move towards OAuth 2.0/OpenID Connect when ready.
   - Integrate with more jurisdictional systems.

Conclusion

By incorporating session ID validation during sign-up and login, and notifying Kutumbua upon successful user migration or login, we achieve:

Enhanced Security:
- Validating the session ID at critical points reduces the risk of unauthorized access.
- Backend notifications ensure Kutumbua is aware of user statuses.

Improved Synchronization:
- Kutumbua maintains up-to-date information on user migrations and logins.
- This facilitates better user management and reporting.

Operational Benefits:
- Ability to track user behavior and migration patterns.
- Enhanced coordination between the new platform and Kutumbua.

Discussion

Q&A

Thank you!

- USE POST FOR USER DATA INSTEAD OF QUERY PARAMS (Otobong)