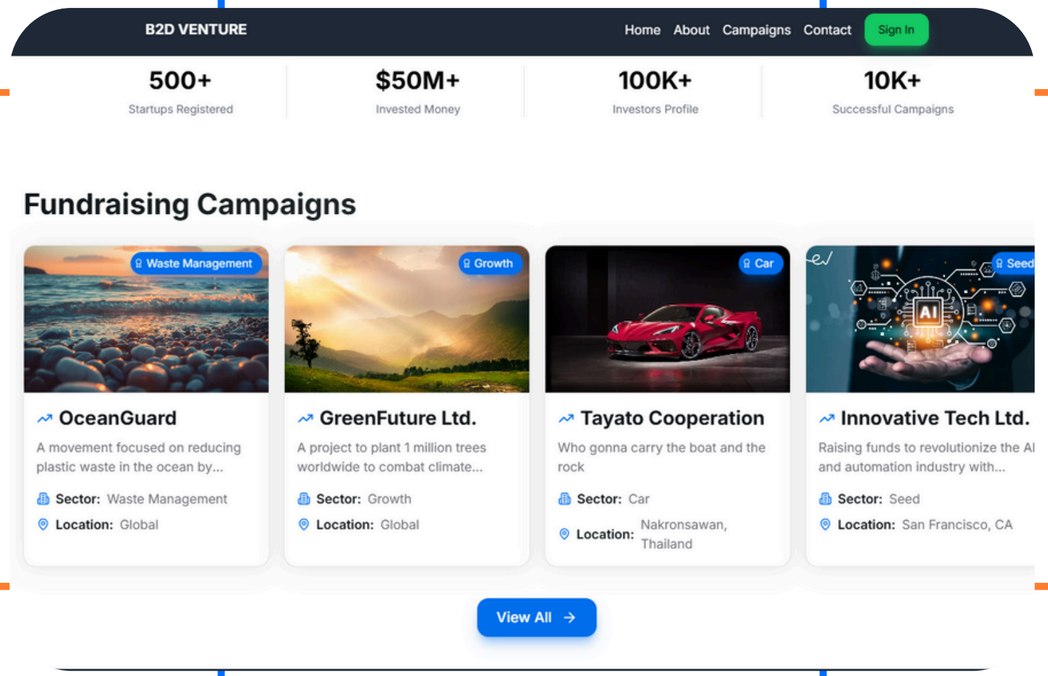
B2D VENTURE SECURITY

PRESENTED BY FISHERMANFRIENDS TEAM

6510545721 WISSARUT KANASUB
6510545349 CHAIYAWUT THENGKET
6510545748 SUKPRACHOKE LEELAPISUTH

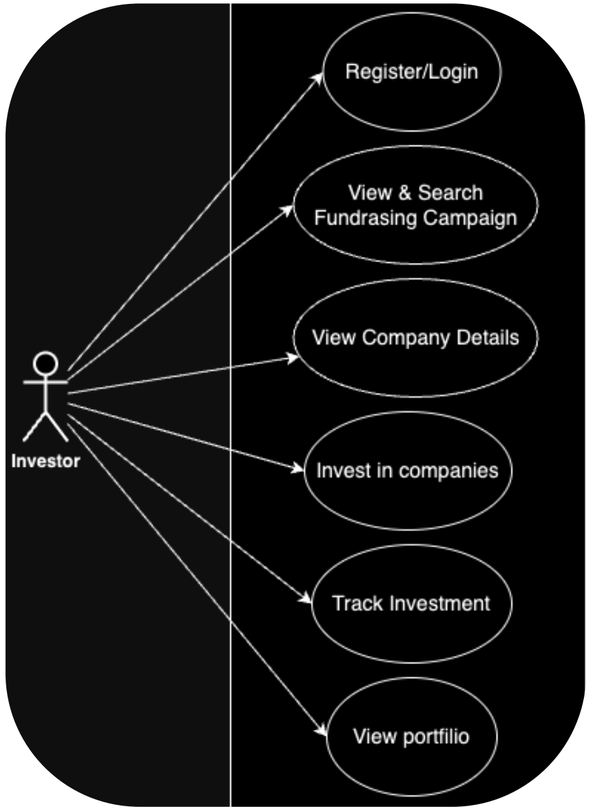
PROJECT

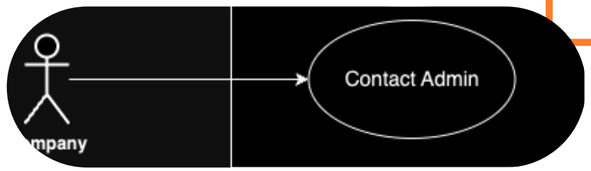
B2D Venture is an investment web application platform that helps companies easily connect with investors. The platform simplifies the investment process while charging a 3% fee on all transactions.

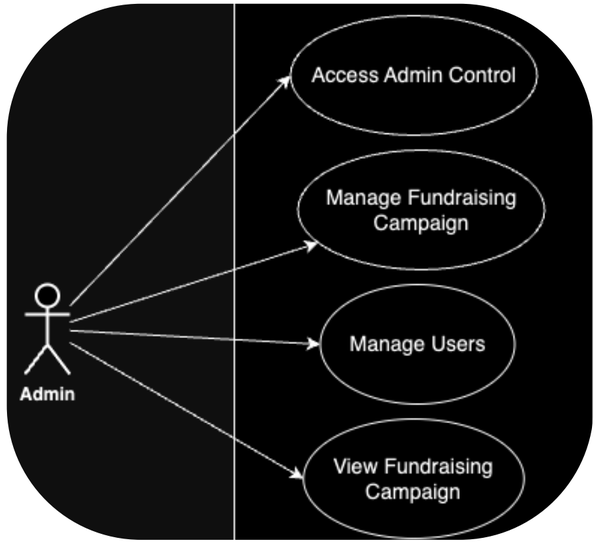


CLIENT REQUIREMENTS

We have developed the web application based on the customer's requirements, transforming them into the agreed-upon use cases







BRIEF USECASE (INVESTOR)

- **Register/Login:** Register an account or log into the system.
- **View & Search Fundraising campaign:** Browse or search for available fundraising opportunities.
- **View Company Details:** Read details about specific companies.
- **Invest in Companies:** Select and invest in a company.
- **Track Investments:** Track the history of investments(Statements).
- **View Portfolio:** View overall investment portfolio.

BRIEF USECASE (ADMIN)

- **Access Admin Control Panel:** Access the control panel to manage fundraising and users.
- **Manage Fundraising Campaign:** Create, update, delete, and view fundraising campaigns.
- **View Users:** View user's account detial.
- **View Fundraising Statements:** View detailed statements for each fundraising campaign including user investments, timestamps, and amounts.

BRIEF USECASE (COMPANY)

Contact Admin: Contact the Admin to provide information and initiate a fundraising campaign ex. email

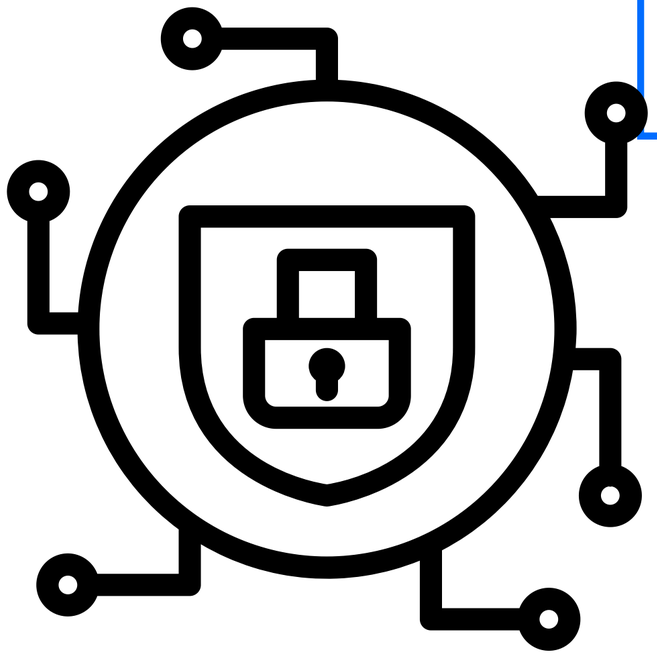
OWASP ASVS SECURITY REQUIREMENTS

Authentication and Session Management

- Register/Login (Auth0)
  - V2.1 Authentication: Ensures secure login using Auth0's Universal Login, aligned with OAuth standards.
  - V3.1 Session Management: Manages session tokens with short expiration times for enhanced security.

Investor Use Cases

- View & Search Fundraising Campaigns / View Company Details
  - V4.3 Access Control: Implements Role-Based Access Control (RBAC) using Auth0 and middleware to restrict access.
  - Invest in Companies / Track Investments / View Portfolio
  - V7.1 Business Logic Security: Enforces secure business logic and authorization processes for investments.
  - V9.2 Data Protection: Encrypts transaction data using HTTPS to secure sensitive information.

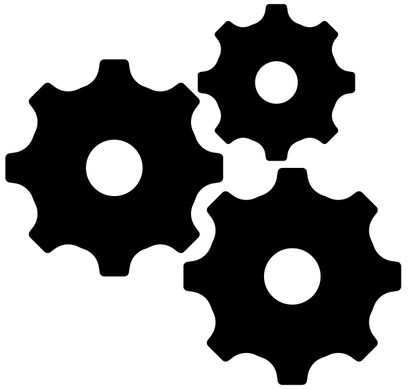


OWASP ASVS SECURITY REQUIREMENTS (2)

- **Admin Use Cases**
  - Access Admin Control Panel / Manage Fundraising Campaigns / View Users / View Fundraising Statements
    - V4.2 Authentication and Authorization Verification: Limits access to admin-only features through middleware.
  - V4.4 Administrative Interface Security: Protects admin interfaces and sensitive features using strict access controls
- **Common Security Features Across the Application**
  - Secure Payment Processing
    - V9.1 Data Protection: Utilizes Stripe to securely manage payment data.
  - GDPR/PDPA Compliance
    - V14.2 Privacy Verification: Encrypts sensitive data and controls data access to comply with privacy regulations.
  - Credential Recovery
  - V2.2 Credential Recovery Verification: Ensures secure account recovery processes to prevent unauthorized access.

OWASP ASVS SECURITY REQUIREMENTS (3)

- **Machine-to-Machine API Protection**
  - V13.2 RESTful Web Services: Secures backend API endpoints using Auth0's M2M authentication to ensure only authorized machines with valid Bearer tokens can access APIs. This includes:
  - Preventing unauthorized access by separating user tokens and M2M tokens.
  - Limiting access to trusted machines or services.
    - Protecting endpoints such as /api/accesstoken with secret keys for secure token requests.



THREAT MODELING USING STRIDE

Identify Core Components and Data Flows

1. **Front-end (Next.js)**: User interface where Investors, Admins, and Companies interact with the application.
2. **Backend (Next.js dynamic routing)**: Handles API requests, data processing, and communication between the front end and database.
3. **Auth0**: Manages user authentication and authorization.
4. **Stripe**: Handles payment processing.
5. **Database (MongoDB)**: Stores user and investment data securely.
6. **Machine-to-Machine API Protection**: Ensures secure communication between trusted services and backend APIs using Bearer tokens.

THREAT MODELING USING STRIDE

1. Spoofing
Definition: An attacker impersonates another user.
Relevant Components: Auth0, Frontend Login
Potential Threats:

- An attacker tries to log in as an Admin or Investor without proper credentials.
- A machine impersonates another trusted machine to access APIs.

Mitigations:

- Enforce strong password policies and monitor unusual login patterns in Auth0.
- Use Machine-to-Machine API Protection to validate Bearer tokens for API requests, ensuring only authorized machines can access backend services.

THREAT MODELING USING STRIDE

2. Tampering

Definition: Unauthorized modification of data.

Relevant Components: Backend API, Database, Machine-to-Machine API Protection

Potential Threats:

- An attacker tampers with investment data or alters user details by modifying API requests.
- An unauthorized machine attempts to modify sensitive data via backend APIs.

Mitigations:

- Validate and sanitize all incoming API requests to prevent malicious payloads.
- Use HTTPS to secure data in transit and prevent tampering during transmission.
- Implement strict **Machine-to-Machine API Protection** to authenticate and authorize requests, ensuring only trusted machines can modify data.

THREAT MODELING USING STRIDE

3. Repudiation
Definition: Denying an action or transaction.
Relevant Components: Backend API, Stripe, MongoDB
Potential Threats:

- A user denies making a specific investment transaction.
- A machine denies submitting API requests that caused a data change.

Mitigations:

- Maintain detailed logging of all actions, with timestamps and user identifiers.

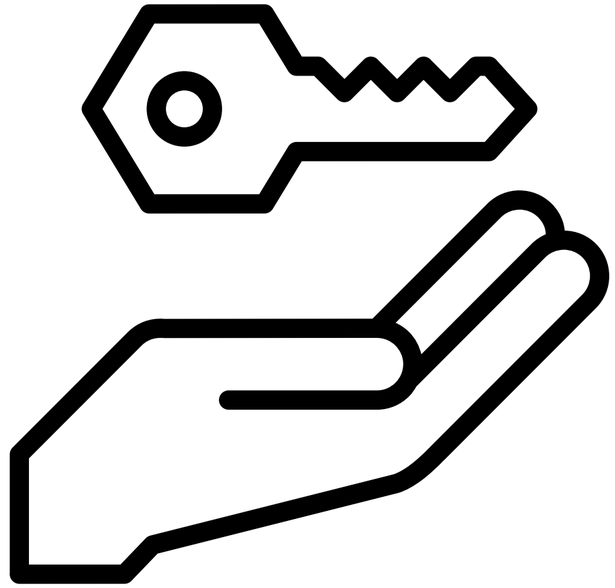
THREAT MODELING USING STRIDE

4. Information Disclosure
Definition: Unauthorized access to sensitive data.
Relevant Components: Database, APIs, Frontend, Machine-to-Machine API Protection
Potential Threats:

- Sensitive data (e.g., investment history, user details) is exposed to unauthorized users or machines.
- An unauthorized machine accesses sensitive API endpoints.

Mitigations:

- Apply Role-Based Access Control (RBAC) to ensure users can only access authorized information.
- Use data encryption for sensitive data stored in MongoDB and enforce HTTPS for all communications.
- Restrict sensitive API endpoints to authorized machines using Machine-to-Machine API Protection.



THREAT MODELING USING STRIDE

5. Denial of Service (DoS)
Definition: Preventing legitimate users from accessing the service.
Relevant Components: Backend API, Frontend, Machine-to-Machine API Protection

6. Elevation of Privilege
Definition: Gaining unauthorized permissions.
Relevant Components: Auth0, Backend APIs, Machine-to-Machine API Protection
Potential Threats:

- An Investor or Company user tries to access Admin functionalities by exploiting authorization flaws.
- A machine with insufficient privileges attempts to access admin-level APIs.

Mitigations:

- Implement strict RBAC policies with Auth0 to restrict access based on user roles.
- Use middleware to enforce role checks on all sensitive routes.
- For machine-to-machine communication, assign specific roles and scopes to tokens, ensuring machines cannot escalate privileges.

SECURITY PRINCIPLES

1. Least Privilege

Users and services have only the minimum access needed for their roles.

Implementation:

- **RBAC:** Distinct roles (Investor, Admin, Company) with appropriate access.
- **Middleware:** Enforces role-based access for routes and APIs.

2. Defense in Depth

Multiple layers of security are in place to protect against various threats.

Implementation:

- **HTTPS & Encryption:** Secure data in transit and protect sensitive information (e.g., payments via Stripe).

3. Fail-Secure

The system defaults to a secure state during failures.

Implementation:

- **Access Denied by Default:** Unauthorized requests are securely rejected.
- **Stripe Error Handling:** Errors notify users without revealing system details

SECURITY PRINCIPLES

4. Separation of Duties

Tasks are divided among roles to prevent misuse or single points of failure.

Implementation:

- **Role Separation:** Admins manage campaigns; Investors initiate transactions.
- **Stripe:** Financial processing handled separately.

5. Secure by Default

Security is prioritized in default configurations.

Implementation:

- **Auth0 Defaults:** Enforce password complexity and token expiration.

6. Minimize Attack Surface

Reduce entry points and exposed features.

Implementation:

- **Sensitive Routes:** Restricted access to authorized machines.

SECURITY PRINCIPLES

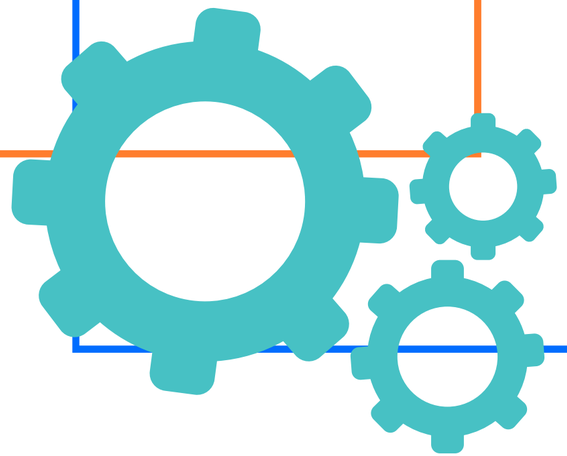
7. Complete Mediation
Every resource access is verified.
Implementation:

- **Auth0 RBAC:** API calls and page access ensure appropriate user roles.
- **Database Controls:** Users only view or modify permitted data.

8. Privacy by Design
Protects user privacy and complies with regulations.
Implementation:

- **Compliance:** GDPR/PDPA standards followed.
- **Data Minimization & Encryption:** Collect only necessary data and encrypt sensitive information.

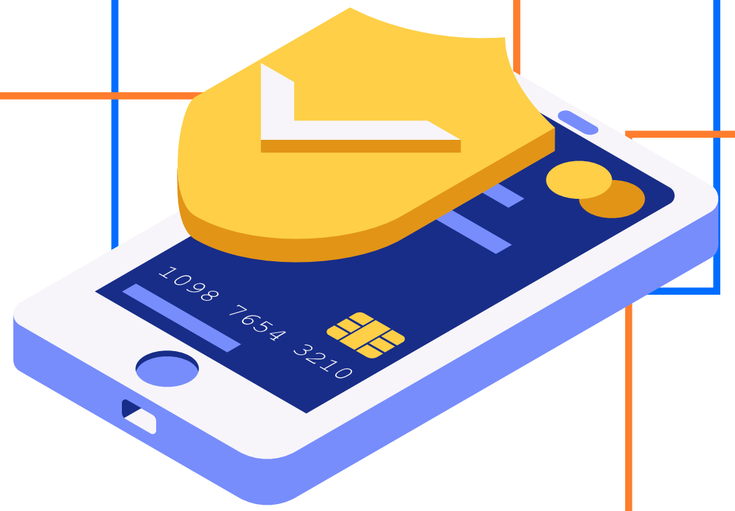
SECURITY FEATURES



Task	Threat Model (STRIDE)	Security Design Principle(s)	Status
Privacy Notification and Consent Form (GDPR/PDPA)	Information Disclosure	Privacy by Design, Least Privilege	Success
Authentication (Login/Logout) and User Management	Spoofing	Authentication, Defense in Depth	Success
Password Masking with Option to Display Plain Text	Information Disclosure	Secure by Default	Success
Password Confirmation	Information Disclosure	Fail-Secure	No develop...
Password Requirements (Length, Characters, etc.)	Spoofing	Least Privilege, Secure by Default	Success
Password Hashing	Information Disclosure	Data Protection, Defense in Depth	Success
Authorization Process with Roles	Elevation of Privilege	Least Privilege, Separation of Duties	Success
Users Can't Access Unauthorized URLs	Elevation of Privilege	Least Privilege, Complete Mediation	Success
Session Management	Spoofing	Session Management, Defense in Depth	Success
CSRF Token	Tampering, Elevation of Privilege	Complete Mediation, Defense in Depth	Unnecessary **reason to support it.**
Secure HTTP Header Settings	Information Disclosure, Spoofing	Secure by Default, Defense in Depth	Success
Random Unique ID Generation	Tampering	Minimize Attack Surface	Success

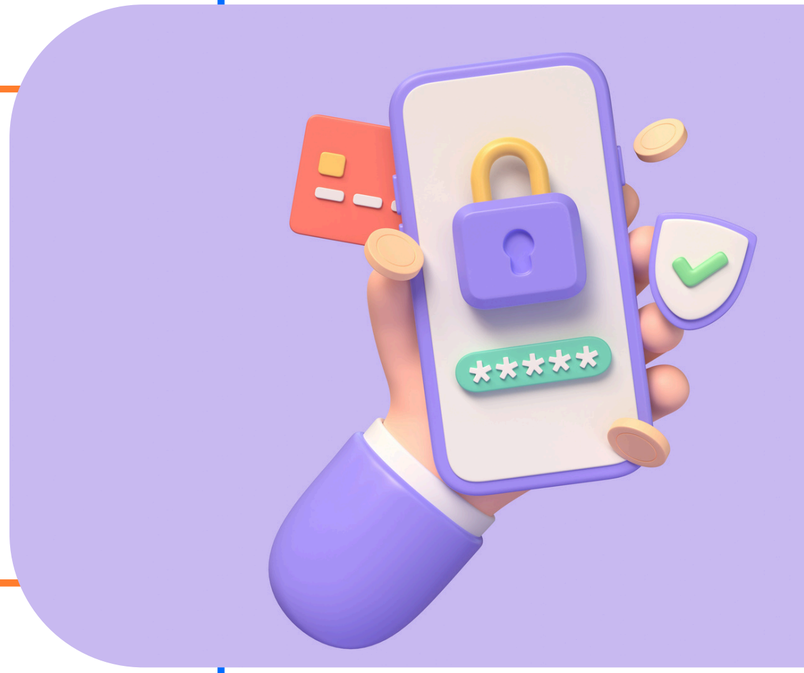
SECURITY FEATURES

Input Validation and SQL Injection Protection	Tampering, Information Disclosure	Defense in Depth, Input Validation	Success
Error Handling	Information Disclosure	Fail-Secure, Defense in Depth	Success
Logging and Monitoring	Repudiation	Complete Mediation, Fail-Secure	Success
Environment Variables	Information Disclosure	Privacy by Design, Secure by Default	Success
Database Encryption	Information Disclosure	Data Protection, Privacy by Design	Success
Security Code Analysis	-	Defense in Depth, Minimize Attack Surface	Success
DAST(Dynamic Application Security Testing) or Pen Test	-	Defense in Depth, Complete Mediation	Success



OAuth/OpenID Social Login	Spoofing, Information Disclosure	Authentication, Privacy by Design	Success
Captcha	Denial of Service, Spoofing	Minimize Attack Surface, Defense in Depth	Success
Login Timeout	Spoofing, Denial of Service	Defense in Depth, Session Management	Success
Forgot Password Feature	Spoofing	Fail-Secure, Authentication	Success

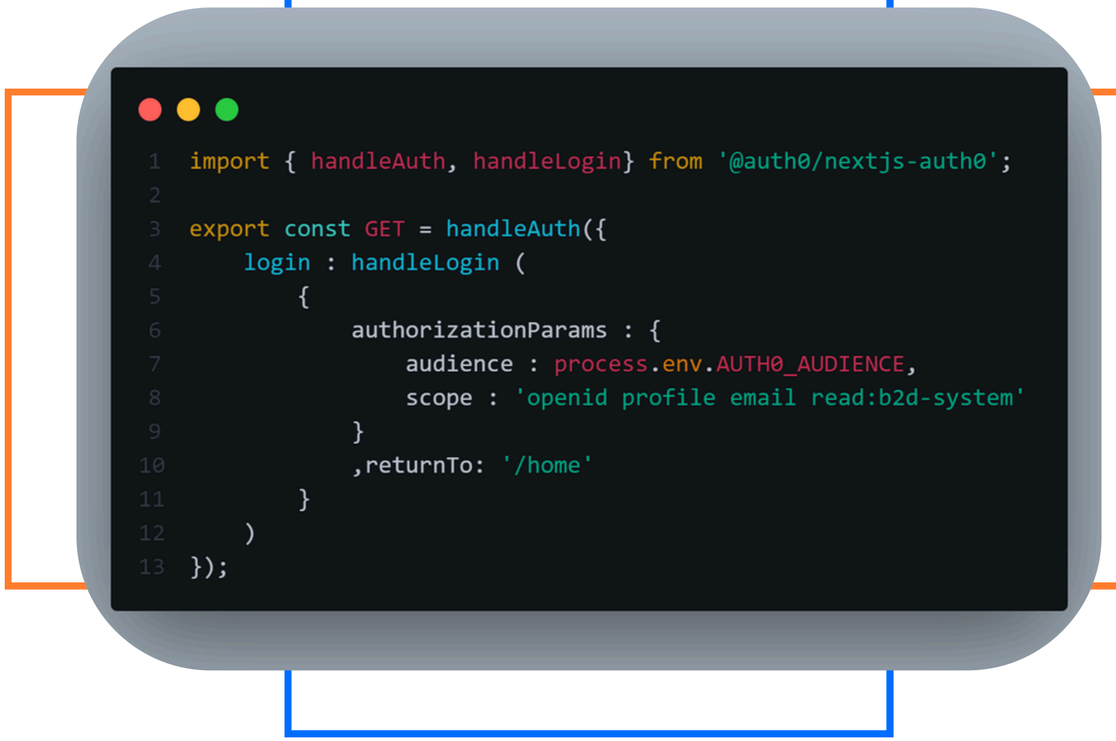
HIGHLIGHT SECURITY FEATURES



AUTHENTICATION (LOGIN/LOGOUT) AND USER MANAGEMENT

Description: Auth0 SDK provide login/ logout page by create/calling an API that Auth0 provideUs

Login: “/auth/login”
logout: “auth/logout”

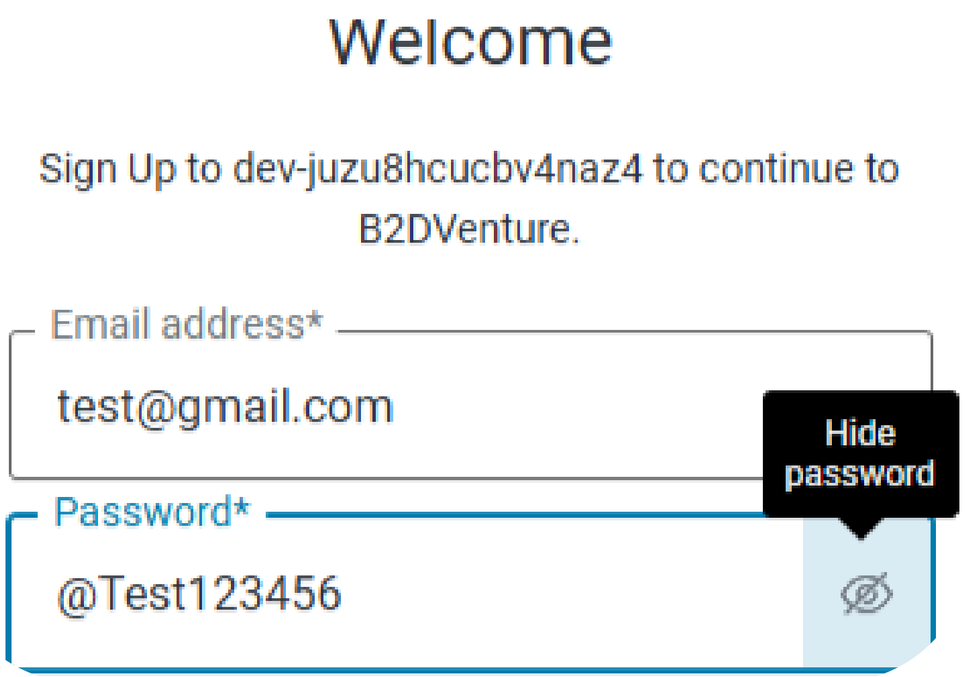


```
import { handleAuth, handleLogin} from '@auth0/nextjs-auth0';

export const GET = handleAuth({
    login : handleLogin (
        {
            authorizationParams : {
                audience : process.env.AUTH0_AUDIENCE,
                scope : 'openid profile email read:b2d-system'
            }
            ,returnTo: '/home'
        }
    )
});
```

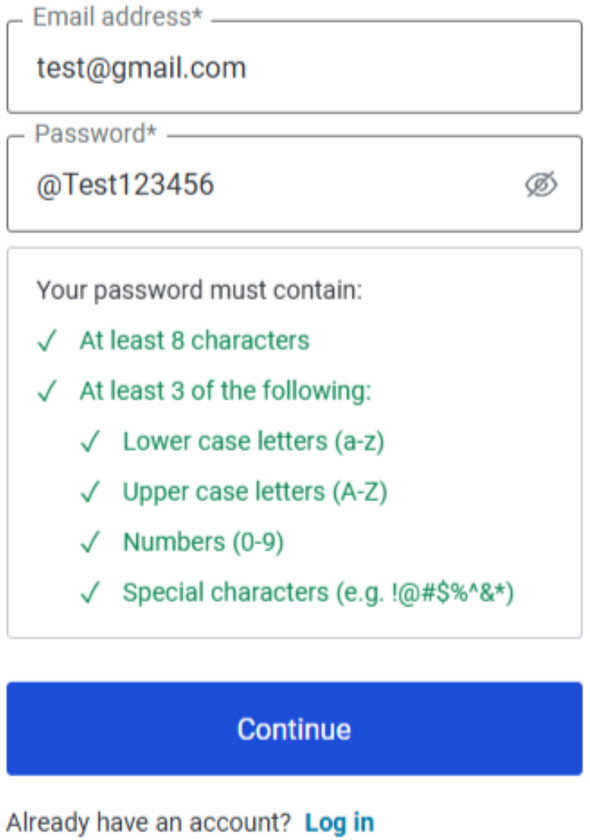
PASSWORD MASKING WITH OPTION TO DISPLAY PLAIN TEXT

Auth0 provide us this services in Universal login



TASK: PASSWORD REQUIREMENTS (LENGTH, CHARACTERS, ETC.)

Auth0 provide us this services in Universal login



PASSWORD HASHING

Auth0 Stores users by them self on their database and hashing users information using high secure algorithm such bcrypt, Argon2

TASK: AUTHORIZATION PROCESS WITH ROLES

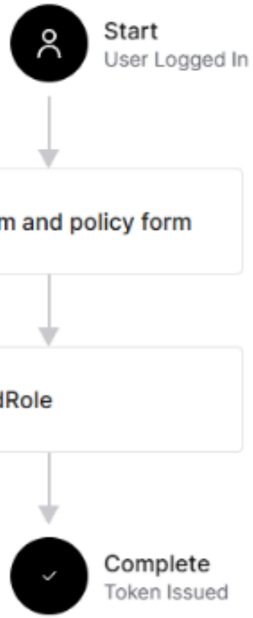
This authorization process defines two roles:

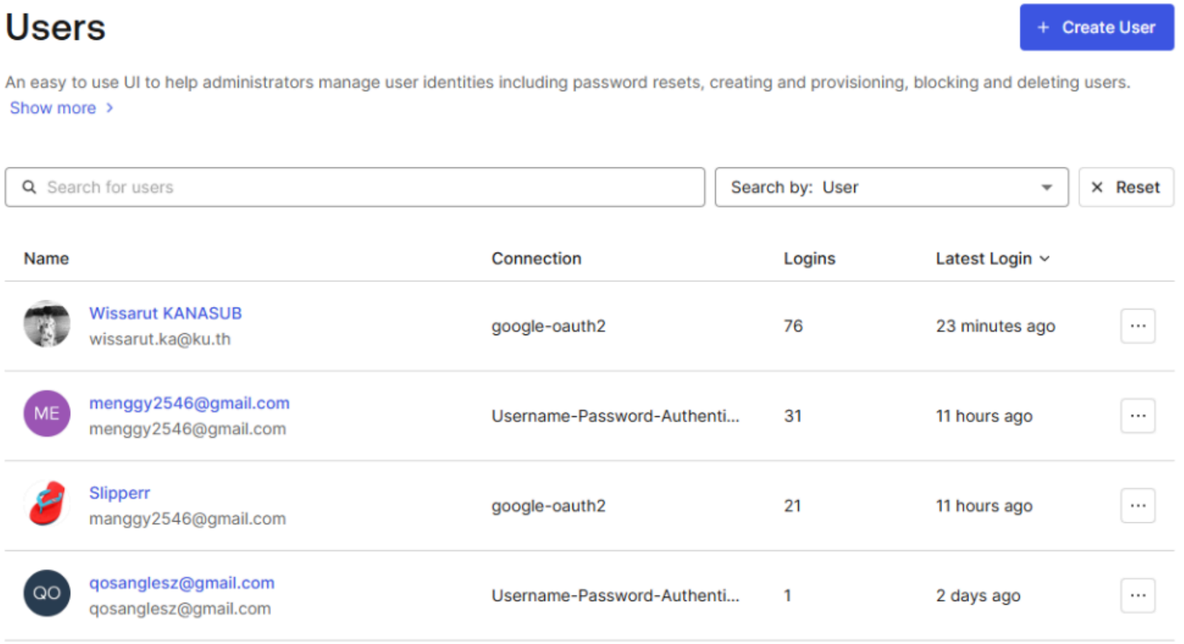
1. **Common User** – Assigned by default to all users upon registration.
2. **Admin** – Manually assigned via the Auth0 Dashboard for users needing admin access.

Setup:

- **Role Assignment:** Default "Common User" role for all users.
- Admin Role: Manually assigned to specific users for admin access.
- **Post Login Flow:** Role is added to the user's session for role-based access control.


Assign Roles
Select roles to assign to this user. You may assign up to 50 roles per user.
Select Roles
Admin B2D
Admin can do everything
User
user can read


Start
User Logged In
Term and policy form
AddRole
Complete
Token Issued


Users
Create User
An easy to use UI to help administrators manage user identities including password resets, creating and provisioning, blocking and deleting users.
Show more
Search for users
Search by: User
Reset
Name
Connection
Logins
Latest Login
Wissarut KANASUB
wissarut.ka@ku.th
google-oauth2
76
23 minutes ago
menggy2546@gmail.com
menggy2546@gmail.com
Username-Password-Authenti...
31
11 hours ago
Slipperr
manggy2546@gmail.com
google-oauth2
21
11 hours ago
qosanglesz@gmail.com
qosanglesz@gmail.com
Username-Password-Authenti...
1
2 days ago

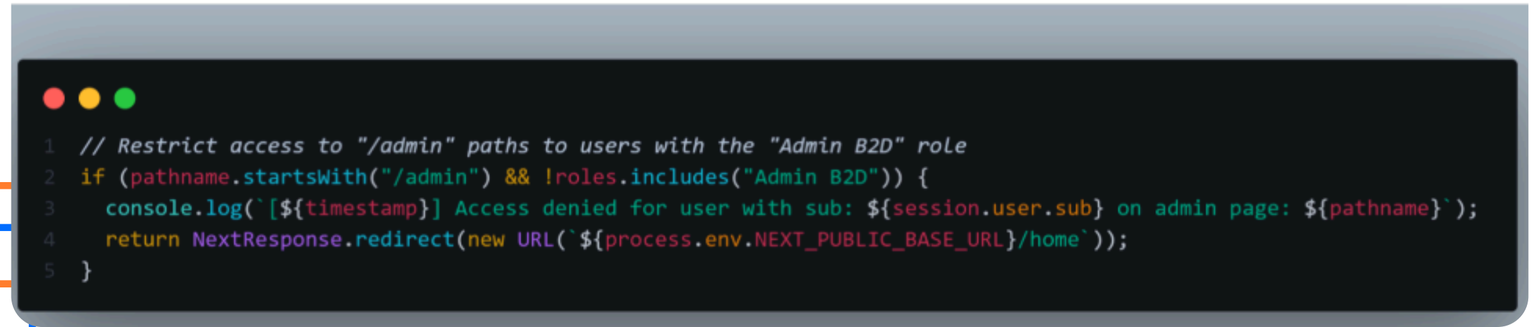
USERS CAN’T ACCESS UNAUTHORIZED URLS

This setup uses the middleware.ts file in Next.js with Auth0's getSession function to enforce authentication and authorization. Here's how it works:

1. **Authentication Check:** Redirects unauthenticated users to the login page.
2. **Role-Based Access Control:** Ensures only users with the “Admin” role can access admin paths like /admin.
3. **User-Specific Data Access Control:** Prevents regular users from accessing or modifying other users' data by verifying the user ID in the URL matches the authenticated user's ID.

-

```
// Redirect unauthenticated users to login page for other paths
if (!session) {
  console.log(`[${timestamp}] Unauthenticated access attempt on: ${pathname}`);
  return NextResponse.redirect(new URL(`${process.env.NEXT_PUBLIC_BASE_URL}/api/auth/login`));
}
```



```
// Restrict access to "/admin" paths to users with the "Admin B2D" role
if (pathname.startsWith("/admin") && !roles.includes("Admin B2D")) {
  console.log(`[${timestamp}] Access denied for user with sub: ${session.user.sub} on admin page: ${pathname}`);
  return NextResponse.redirect(new URL(`${process.env.NEXT_PUBLIC_BASE_URL}/home`));
}
```

INPUT VALIDATION AND SQL INJECTION PROTECTION

The web application, built with Next.js and MongoDB, is secured against NoSQL injection through the following measures:

1. **UUIDv4 Validation:** Ensures campaign IDs follow the correct format to prevent malformed IDs.
2. **Input Sanitization:** Removes sensitive fields like id and _id to avoid accidental key overwriting.
3. **Required Fields Validation:** Checks essential fields (e.g., companyName, website) for completeness before data insertion or update.
4. **NoSQL Injection Protection:** Validates and sanitizes inputs to prevent malicious data from being executed in MongoDB queries.

```
async delete(id: string): Promise<boolean> {
        if (!CampaignRepository.validateUuid(id)) {
            throw new Error('Invalid campaign ID');
        }

        const collection = await this.getCollection();
        const result = await collection.deleteOne({ id: id
        return result.deletedCount > 0;
    }
```

```
async update(id: string, updatedData: Partial<Campaign>): Promise<boolean> {
    if (!CampaignRepository.validateUuid(id)) {
        throw new Error('Invalid campaign ID');
    }

    // Sanitize the updated data (remove malicious fields)
    const sanitizedData = { ...updatedData };
    delete sanitizedData._id; // Remove _id field to avoid overwriting the original ID
    delete sanitizedData.id; // Remove id field if it exists

    const collection = await this.getCollection();
    const result = await collection.updateOne({ id: id }, { $set: sanitizedData });
    return result.matchedCount > 0;
}
```

CampaignRepository.ts

```
// Validate if the ID is a valid UUIDv4
private static validateUuid(id: string): boolean {
    const uuidPattern = /^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-4[0-9a-fA-F]{3}-[89aAbB][0-9a-fA-F]{3}-[0-9a-fA-F]{12}$/;
    return uuidPattern.test(id);
}
```

SECURE HTTP HEADER SETTING

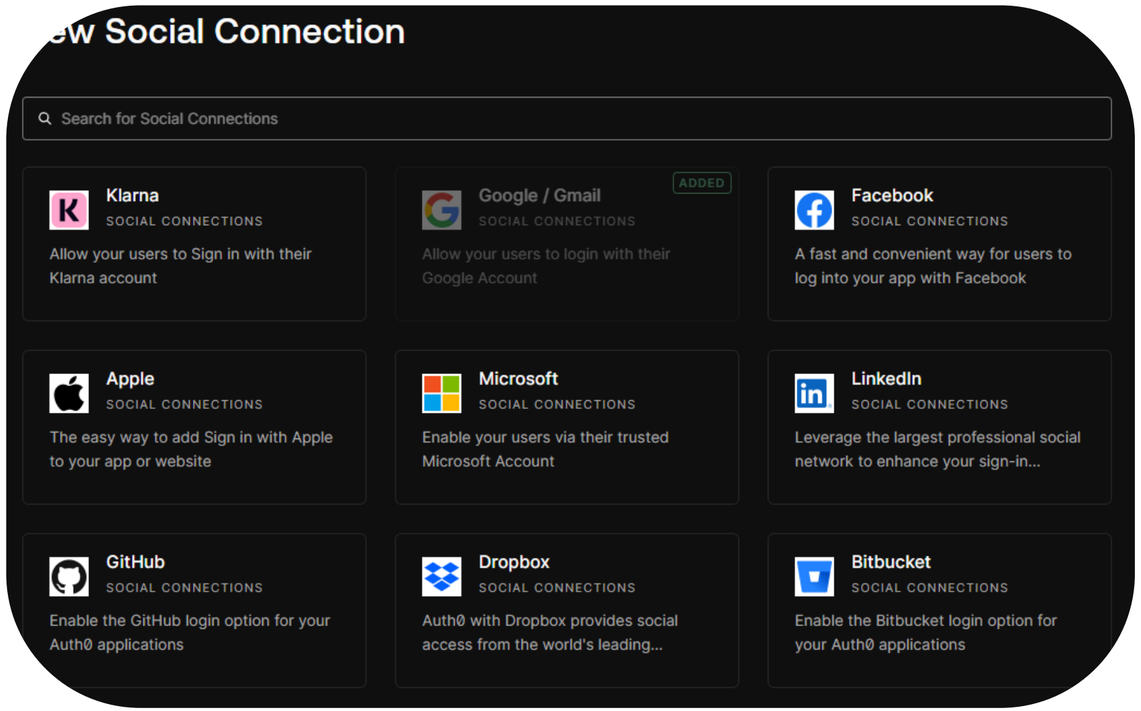
Actions:

1. **Strict-Transport-Security (HSTS):** Enforces HTTPS communication, preventing MITM attacks.
2. **Content-Security-Policy (CSP):** Restricts content sources to prevent XSS attacks.
3. **X-Content-Type-Options:** Prevents MIME type sniffing by browsers.
4. **X-Frame-Options:** Blocks clickjacking by preventing the app from being embedded in iframes.
5. **X-XSS-Protection:** Activates the browser's XSS filter to block attacks.
6. **Referrer-Policy:** Limits the exposure of referrer information during navigation.

```
response.headers.set('Strict-Transport-Security',
'max-age=31536000; includeSubDomains; preload');
  response.headers.set('Content-Security-Policy', "default-src
'self'; script-src 'self'; style-src 'self'; img-src 'self';
font-src 'self'; connect-src 'self';");
  response.headers.set('X-Content-Type-Options', 'nosniff');
  response.headers.set('X-Frame-Options', 'DENY');
  response.headers.set('X-XSS-Protection', '1; mode=block');
  response.headers.set('Referrer-Policy',
'no-referrer-when-downgrade');
```

OAUTH/OPENID SOCIAL LOGIN

Auth0 Provide many social media login but in this project we use google authentication



DATABASE ENCRYPTION

The web application uses MongoDB Atlas for database encryption, which provides Encryption at Rest by default. All data in MongoDB Atlas clusters is encrypted using AES-256 encryption without requiring manual configuration. This ensures sensitive data is secure, and no additional development work is needed for encryption, as it is automatically handled by MongoDB Atlas.

Encryption at Rest using Customer Key Management

IMPORTANT

Feature unavailable in Serverless Instances

Serverless instances don't support this feature at this time. To learn more, see Serverless Instance Limitations.

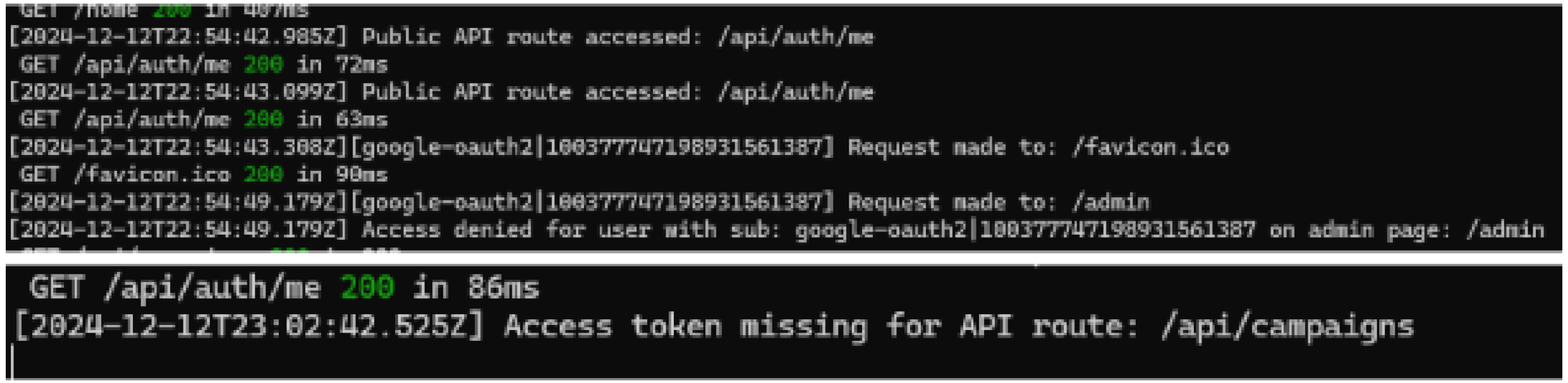
Atlas encrypts all cluster storage and snapshot volumes at rest by default. You can add another layer of security by using your cloud provider's KMS together with the MongoDB encrypted storage engine.

Configuring Encryption at Rest using your Key Management incurs additional charges for the Atlas project. To learn more, see Advanced Security.

LOGGING MONITORING & ERROR HANDLING

This project implements logging and error handling for both server-side and client-side using **Middleware.ts**:

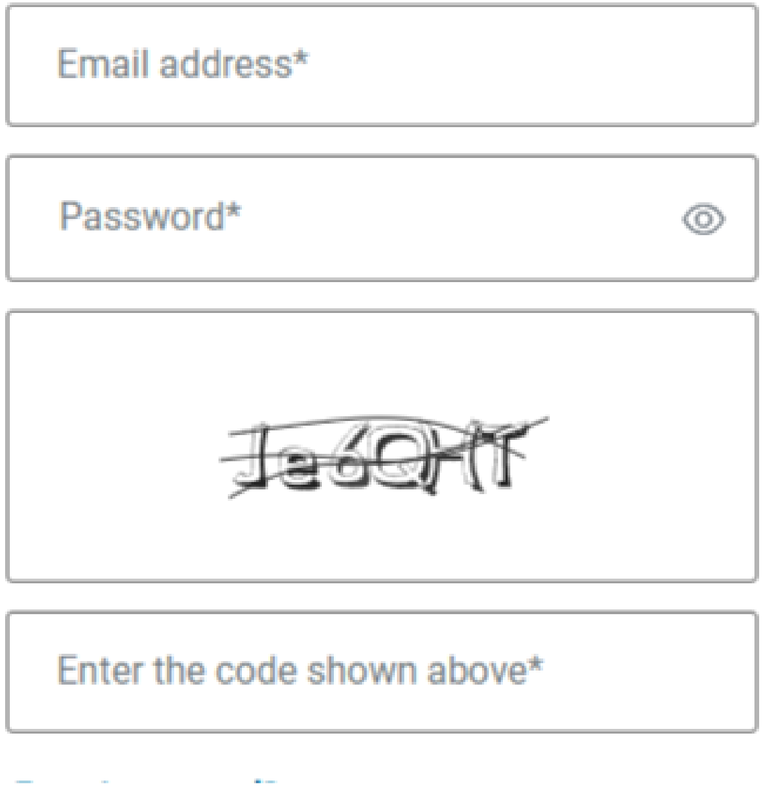
- **Server-Side (SSR) Logging:**
  - console.log() tracks access attempts and errors, logging details like user context and timestamps.
- **Client-Side Error Handling:**
  - Errors are caught in try-catch blocks, with user-friendly error messages displayed on the screen.



CAPTCHA

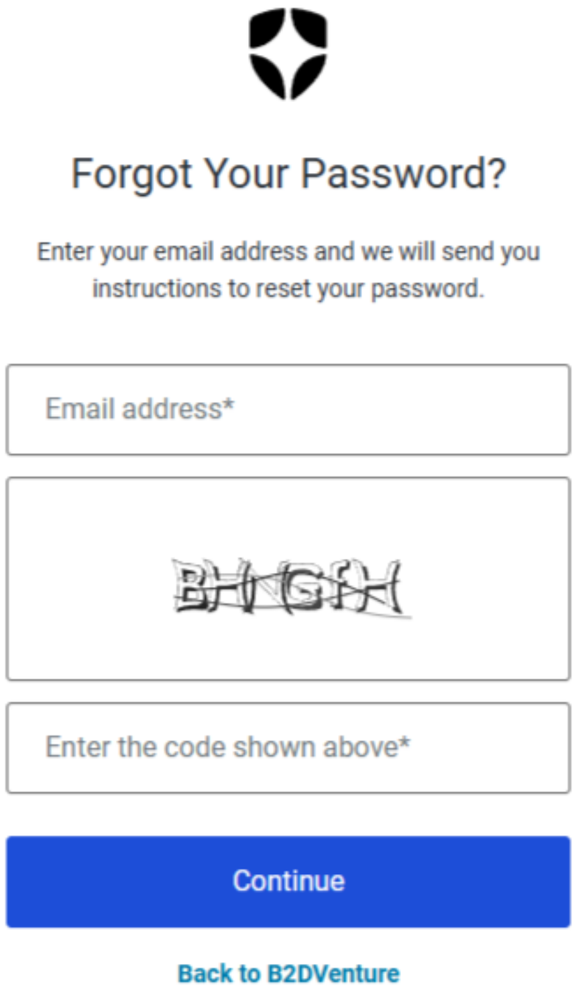
Description: In this project, we integrate Captcha functionality using Auth0's built-in service. This Captcha is triggered during key authentication processes to prevent automated bot activity.

- **Triggering Events:**
  - **Login:** Captcha will appear during the login process to ensure the user is human.
  - **Sign Up:** Captcha will also be required during user sign-up to prevent automated account creation.
  - **Forgot Password:** If a user requests a password reset, Captcha will be triggered to avoid automated password reset requests.



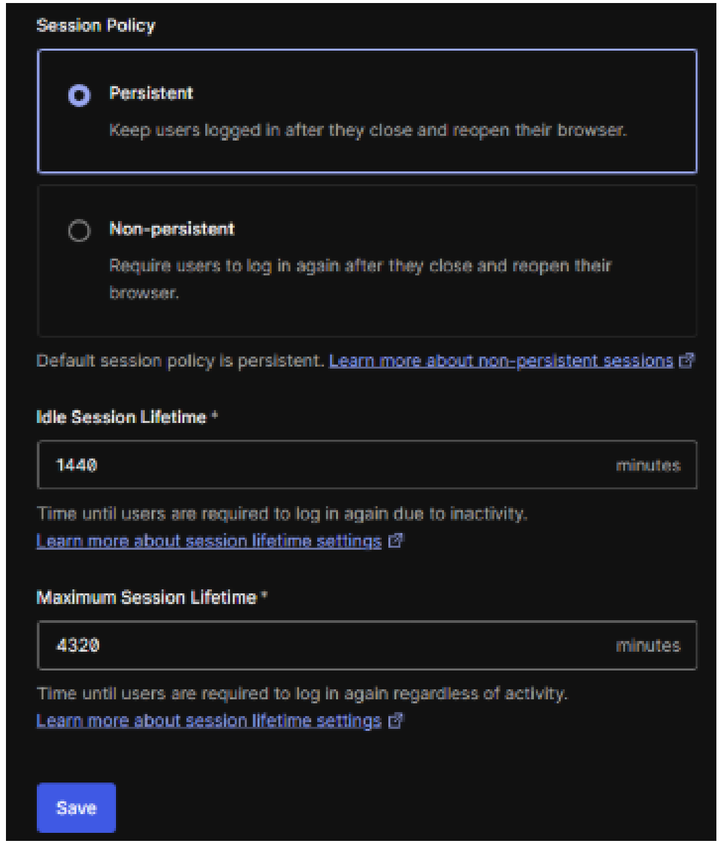
FORGOT PASSWORD FEATURE

- **Description:** The Forgot Password feature is implemented using Auth0's built-in service, which simplifies the process of password recovery and ensures secure handling of user credentials.



LOGIN TIMEOUT

- **Description:** The Login Timeout feature is managed through Auth0 by configuring session expiration settings.
  - **Idle Timeout:** The session will expire after 1 day of inactivity.
  - **Maximum Session Duration:** The session will not last longer than 3 days, ensuring users must re-authenticate after this period, enhancing security.



BACKEND API AUTHENTICATION WITH TOKENS

For securing backend API endpoints, we use Auth0's Machine-to-Machine (M2M) API protection, requiring a valid Bearer token for access.

- **Bearer Token Requirement:** Backend endpoints are protected by a Bearer token, ensuring only authenticated machines (like the frontend) can access them.
- **User and Role Separation:** Tokens are separate for user authentication and roles, enforcing role-based access control on frontend pages.
- **Public Endpoints:** Some endpoints allow open access, but sensitive ones remain protected.
- **Special Token Endpoint:** The /api/accesstoken endpoint allows token requests from Auth0, requiring a secret key for authorization.
    -

MIDDLEWARE.TS

```ts
// Validate API key for `/api/accesstoken` route
if (pathname.startsWith('/api/accesstoken')) {
  const apiKey = request.headers.get('accesstokenapikey');
  if (apiKey !== process.env.NEXT_PUBLIC_ACCESS_TOKEN_API_KEY) {
    console.error(`[${timestamp}] Unauthorized access attempt on: ${pathname} with invalid API key`);
    return NextResponse.json({ error: 'Unauthorized: Invalid or missing API key' }, { status: 401 });
  }
  return NextResponse.next();
}
```

```ts
// Validate Authorization header for other API routes
const token = request.headers.get('Authorization')?.split(' ')[1];
if (!token) {
  console.error(`[${timestamp}] Access token missing for API route: ${pathname}`);
  return NextResponse.json({ error: 'Access token is required' }, { status: 401 });
}
```

```ts
// Validate access token issuer and audience
try {
  const decoded = decodeJwt(token);
  if (decoded.iss === process.env.AUTH0_ISSUER_BASE_URL && decoded.aud === process.env.AUTH0_CLIENT_ID) {
    console.log(`[${timestamp}] Access token valid for user with sub: ${decoded.sub} for API route: ${pathname}`);
    return NextResponse.next();
  }
} catch (error) {
  console.error(`[${timestamp}] Invalid or expired access token for API route: ${pathname}`);
  return NextResponse.json({ error: 'Invalid or expired access token' }, { status: 403 });
}
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "error": "Access token is required"
3  }
```

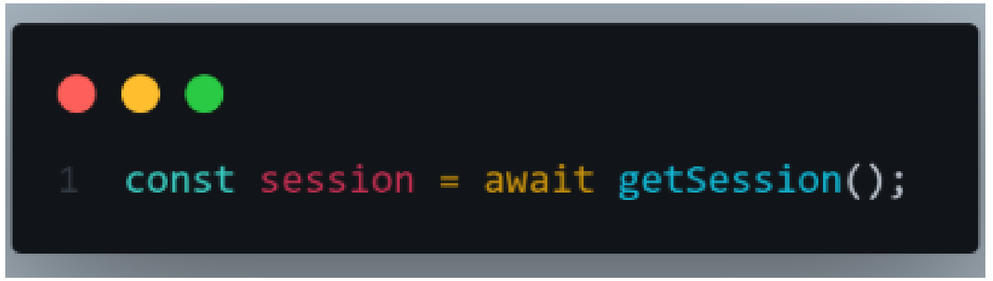
CSRF TOKEN

With Auth0's Machine-to-Machine (M2M) API protection, CSRF tokens are unnecessary because:

- **M2M Communication:** Communication happens between trusted machines, not browsers, eliminating CSRF risks.
- **Bearer Tokens:** M2M uses Bearer tokens (e.g., JWT) in the Authorization header for authentication, instead of cookies.
- **No User Interaction:** Since there's no user interaction in M2M, CSRF attacks don't apply.

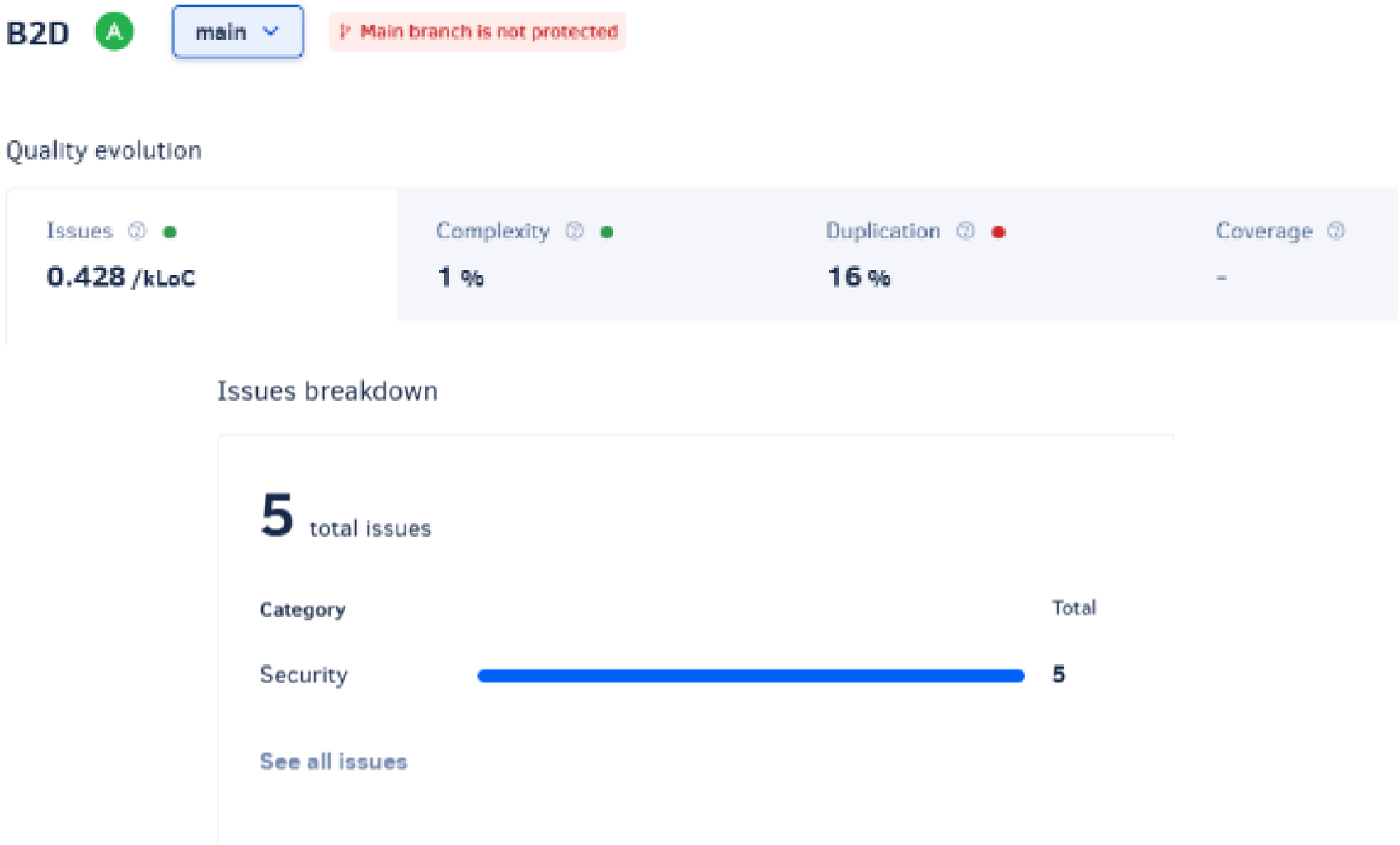
SESSION MANAGEMENT

The Auth0 SDK simplifies session management by using the api/auth/me endpoint to retrieve the user's session. With just one line of code, session state is automatically managed, eliminating the need for manual session handling and ensuring seamless authentication.



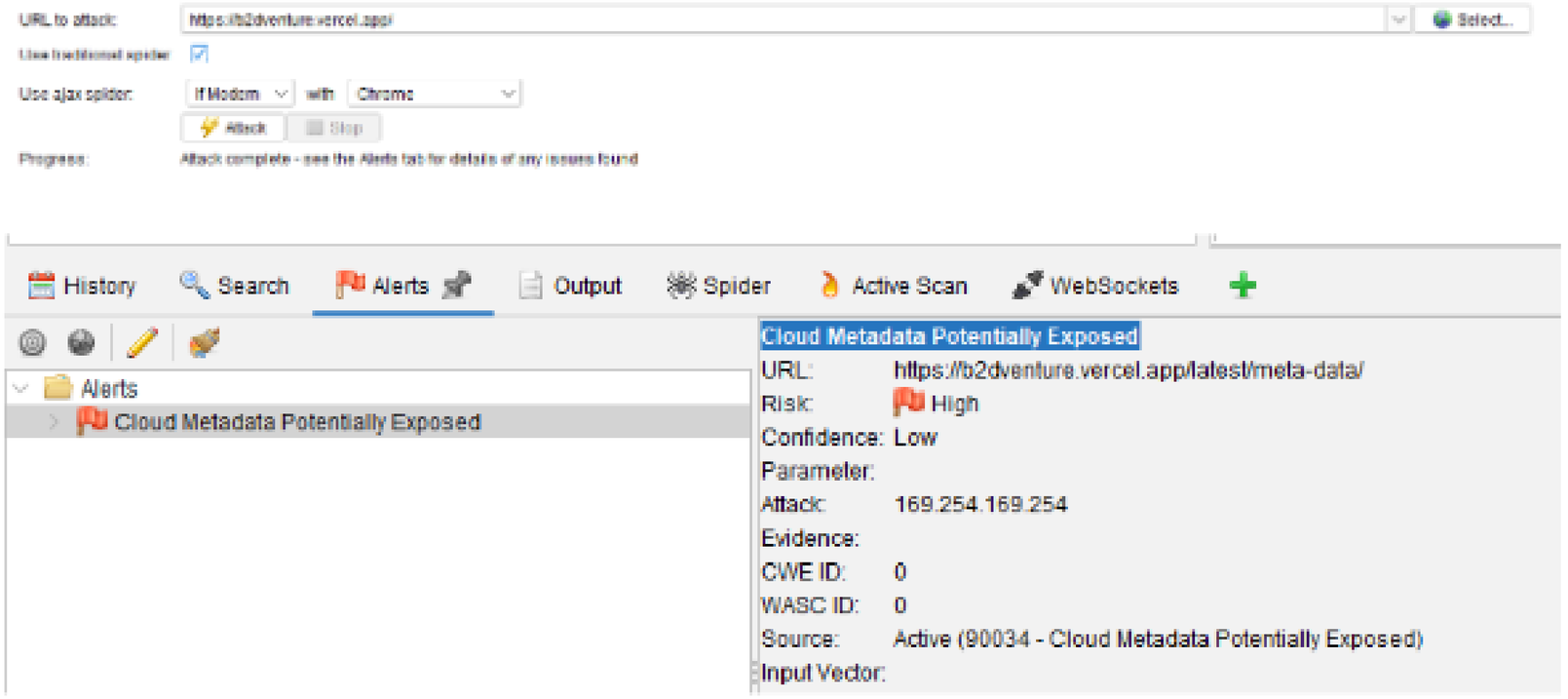
SECURITY CODE ANALYSIS

Currently, 5 security issues have been identified due to outdated dependencies that are no longer supported or maintained. But other implementation has secured



DYNAMIC APPLICATION SECURITY TESTING

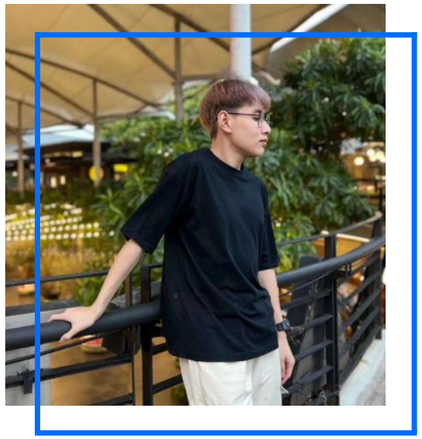
We ran an OWASP ZAP scan on https://b2dventure.vercel.app/ and found a "Cloud Metadata Potentially Exposed" alert, typically caused by misconfigured NGINX servers. Since the app is hosted on Vercel, which doesn't use NGINX, this is not a concern. Vercel's infrastructure provides adequate protection.



DEMO

THANK YOU

THE TEAM



WISSARUT KANASUB

Full-stack developer

CHAIYAWUT THENGKET

Full-stack developer

SUKPRACHOKE LEELAPISUTH

Software Tester