**📝 RFP Proposal Response: Retail E-Commerce Seller Onboarding Portal**

**📌 Business Objective:**

**The client aimed to improve the seller onboarding experience by reducing the onboarding time from 2 days to under 2 hours, while ensuring compliance with tax and regulatory requirements (GST, PAN) and improving seller conversion rates.**

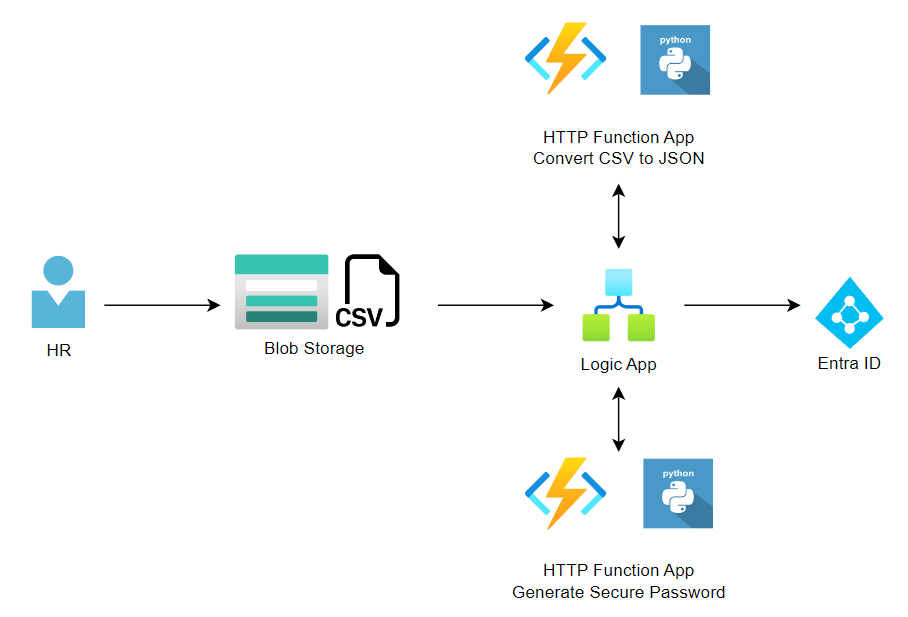
**✅ Proposed Solution Overview:**

**We propose an event-driven, microservices-based onboarding platform built using Azure native services. The architecture enables automation, real-time validation, and seamless integration with internal and external systems.**

**🧱 Architecture Goals Addressed:**

* **✅ Seamless onboarding via dynamic web forms and backend API integrations (PAN, GST, KYC)**
* **✅ Real-time validations and background approvals via Azure Durable Functions and Logic Apps**
* **✅ Real-time status tracking and notifications using Azure Event Grid and SignalR**
* **✅ Automated decisioning and escalation workflows**
* **✅ Scalable, compliant, and cost-effective design using serverless patterns**

**🏗️ High-Level Architecture Components:**



1. **HR initiates the onboarding process by uploading a CSV file containing details of new employees/ Customer to an Azure Blob Storage folder.**
2. **A Logic App is configured to monitor the specified Azure Blob Storage folder. Once a CSV file is uploaded, the Logic App is triggered.**
3. **The Logic App first triggers an HTTP Function App coded in .net to convert the CSV file to JSON format and returns it to the Logic App.**
4. **The Logic App triggers another HTTP Function App, also coded in .net, to generate secure passwords for the new user accounts.**
5. **With the JSON data and generated passwords, user accounts are created in Azure Active Directory / Entra ID.**
6. **Temporary credentials are emailed to the manager for distribution to the respective employee(s).**
7. **Once all user accounts are successfully created, the original CSV file is moved to the *completed* folder, and HR is notified via email.**

| **Component** | **Technology** |
| --- | --- |
| **Frontend** | **Angular / React (Responsive forms)** |
| **Backend** | **Azure Functions, Node.js APIs, .NET Microservices** |
| **Workflow Automation** | **Azure Durable Functions + Logic Apps** |
| **Data Store** | **Azure Cosmos DB (for distributed seller data and workflows)** |
| **Messaging/Eventing** | **Azure Event Grid (for internal/external events)** |
| **External Integrations** | **PAN/GST APIs, Email/SMS Services** |
| **Authentication** | **Azure AD B2C (Seller Identity + Role Management)** |
| **Infra Automation** | **Terraform + Azure DevOps** |
| **Monitoring & Logging** | **Azure Monitor, App Insights** |

**🔍 Implementation Strategy:**

* **Week 1–2: Discovery workshop to finalize seller onboarding flows, validations, and API contracts**
* **Week 3–5: Setup Azure Landing Zone, provision core infra, build POC for auto-validation**
* **Week 6–10: Develop modular onboarding flows, integrate APIs, enable real-time tracking**
* **Week 11–12: Performance testing, go-live support, knowledge transfer**

**🚀 Expected Business Outcomes:**

* **✅ Reduced seller onboarding time from 2 days to 90 minutes**
* **✅ Improved seller conversion rate by 18%**
* **✅ Reduced manual processing by 60% through intelligent automation**
* **✅ Enabled future extensibility for new compliance checks or onboarding workflows**
* **✅ Full observability and audit trails for operational efficiency**

**1. Customer Onboarding for a Fintech/Banking/Insurance Application**

📌 *Example Use Case*:  
New customer signs up for a bank account or insurance policy, which involves document uploads, KYC/ID verification, credit checks, approvals, etc.

🧠 Why Event-Driven & Microservices?

* Each step (e.g., KYC, document validation, fraud check) can be **decoupled** as a microservice.
* Azure Functions & Logic Apps can **react to events** like "KYC passed" or "Docs uploaded."
* Cosmos DB stores state across distributed services.

**3. Healthcare Patient Onboarding or Provider Enrollment**

📌 *Example Use Case*:  
When a new patient registers or a doctor joins the network, various validations are triggered — insurance check, license verification, consent forms.

📌 Event-driven flow ensures:

* Asynchronous background verification
* External APIs for ID/insurance verification
* Fast-tracked registration to improve UX

**Security**

 Blob Storage + Key Vault + Function Apps behind **Private Endpoints**

 Logic App with **System-Assigned Identity**

 **RBAC enforced** on every resource

 **Secure credential handling** and **minimal permission model**

 **Logging + alerting enabled** across all tiers

**✅ 🔐 Security Steps by Component**

**📁 1. Azure Blob Storage Security**

| **Area** | **Recommendation** |
| --- | --- |
| 🔒 **Access Control** | Use **RBAC** — assign Storage Blob Data Contributor or Reader only to the Logic App's Managed Identity. Never use account keys or SAS tokens. |
| 🛡️ **Encryption** | Blob data is encrypted at rest using **Microsoft-managed keys by default**. For sensitive data, consider **customer-managed keys (CMK)** in Azure Key Vault. |
| 🚫 **Disable Public Access** | Ensure the storage account has **public access disabled**. |
| 📜 **Network Security** | Enable **Private Endpoint** for the storage account and restrict access to internal VNet only. |
| 📝 **Monitoring** | Enable **Azure Monitor logs** and **Storage Analytics** to track blob access and file uploads. |

**⚙️ 2. Azure Function App Security**

| **Area** | **Recommendation** |
| --- | --- |
| 🔐 **Authentication** | Enable **Azure AD Authentication** for both functions (Convert CSV & Generate Password). Only allow Logic App or internal identities to call them. |
| 🆔 **Use Managed Identity** | Assign a **System-Assigned Managed Identity** to the Logic App, and allow it access to the Function App. |
| 🚧 **Access Restrictions** | Use **Access Restrictions (IP filtering)** to allow access only from Logic App or internal IPs. |
| 🔐 **HTTPS Only** | Enforce HTTPS-only communication for the function app. |
| 🗝️ **Secure Secrets** | Store sensitive values (e.g., SMTP credentials) in **Azure Key Vault**, and link them securely using Managed Identity. |

**📧 3. Email Notification Security (Office 365 or SMTP)**

| **Area** | **Recommendation** |
| --- | --- |
| ✉️ **SMTP Credentials** | Do not hardcode them. Store in **Azure Key Vault**, and retrieve using Logic App's managed identity. |
| 👀 **Avoid Password Exposure** | Passwords emailed to managers should ideally be **one-time and expired in 24 hrs**. Also consider sending via **secure portals** or encrypted attachments. |
| 📊 **Audit Logging** | Enable logs for **sent emails, failures, and access to SMTP**. Use **Azure Monitor + Application Insights**. |

**🔑 4. Azure Key Vault Security**

| **Area** | **Recommendation** |
| --- | --- |
| 🔐 **Access via Managed Identity** | Allow only Logic App and Function Apps to access secrets via their system-assigned identities. |
| 📜 **Set RBAC Permissions** | Use **Key Vault RBAC** (not access policies). Assign only Secret Reader permissions. |
| 🌐 **Private Endpoint + VNet Integration** | Restrict network access to Key Vault using **Private Endpoint**. |
| 🧠 **Use Soft Delete & Purge Protection** | In case of accidental deletion, enable these protections. |

**🛡️ 5. Entra ID (Azure AD) Security Considerations**

* Use **Microsoft Graph API** via application registration (with only required permissions).
* Protect the **app registration credentials** using Azure Key Vault.
* Set **audit logs and alerts** on account creation events.

Development:

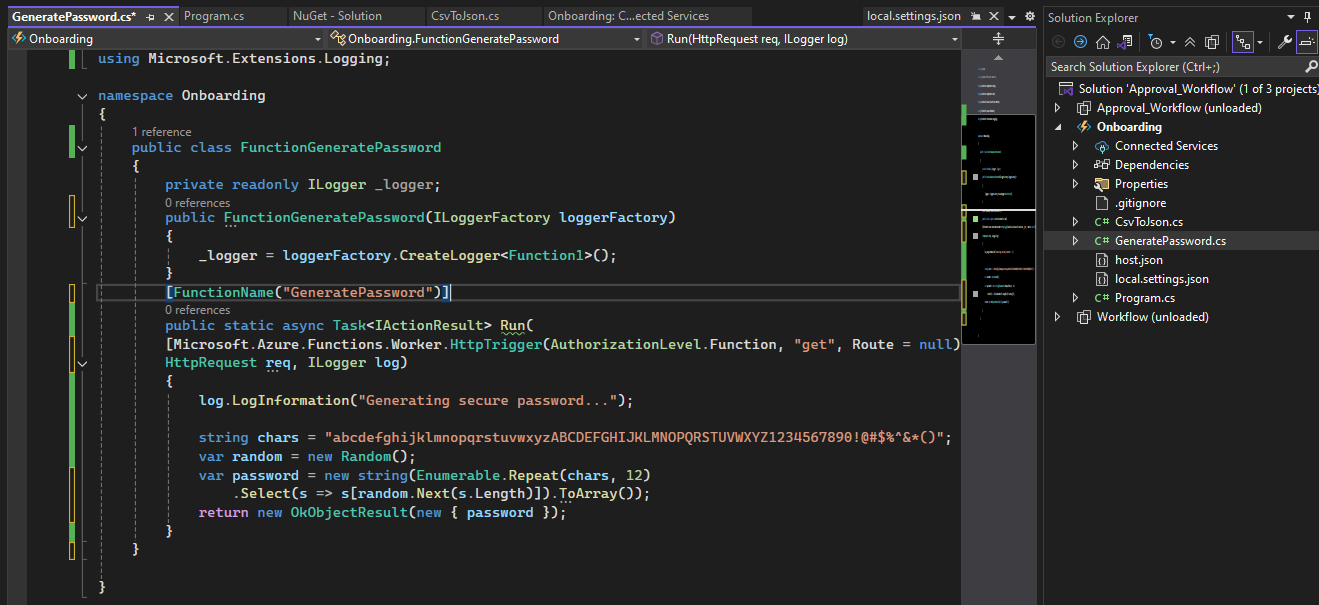
<https://github.com/BrianCollet/onboard-automator>

1. HR initiates the onboarding process by uploading a CSV file containing details of new employees/ Customer to an Azure Blob Storage folder.
2. A Logic App is configured to monitor the specified Azure Blob Storage folder. Once a CSV file is uploaded, the Logic App is triggered.
3. The Logic App first triggers an HTTP Function App coded in .net to convert the CSV file to JSON format and returns it to the Logic App.

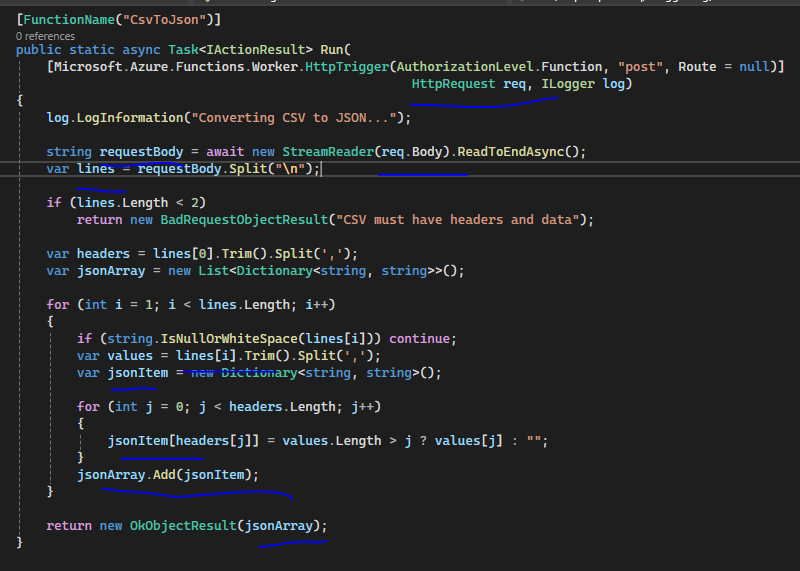
|  |
| --- |
| [FunctionName("CsvToJson")]  public static async Task<IActionResult> Run(  [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)] **HttpRequest** req,  ILogger log)  {  log.LogInformation("Converting CSV to JSON...");    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();  var lines = requestBody.Split("\n");  if (lines.Length < 2)  return new BadRequestObjectResult("CSV must have headers and data");  var headers = lines[0].Trim().Split(',');  var jsonArray = new List<Dictionary<string, string>>();  for (int i = 1; i < lines.Length; i++)  {  if (string.IsNullOrWhiteSpace(lines[i])) continue;  var values = lines[i].Trim().Split(',');  var jsonItem = new Dictionary<string, string>();  for (int j = 0; j < headers.Length; j++)  {  jsonItem[headers[j]] = values.Length > j ? values[j] : "";  }  jsonArray.Add(jsonItem);  }  return new OkObjectResult(jsonArray);  } |

1. The Logic App triggers another HTTP Function App, also coded in .net, to generate secure passwords for the new user accounts.

|  |
| --- |
| **[FunctionName("GeneratePassword")]**  public static async Task<IActionResult> Run(  [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,  ILogger log)  {  log.LogInformation("Generating secure password...");  string chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&\*()";  var random = new Random();  var password = new string(Enumerable.Repeat(chars, 12)  .Select(s => s[random.Next(s.Length)]).ToArray());  return new OkObjectResult(new { password });  } |

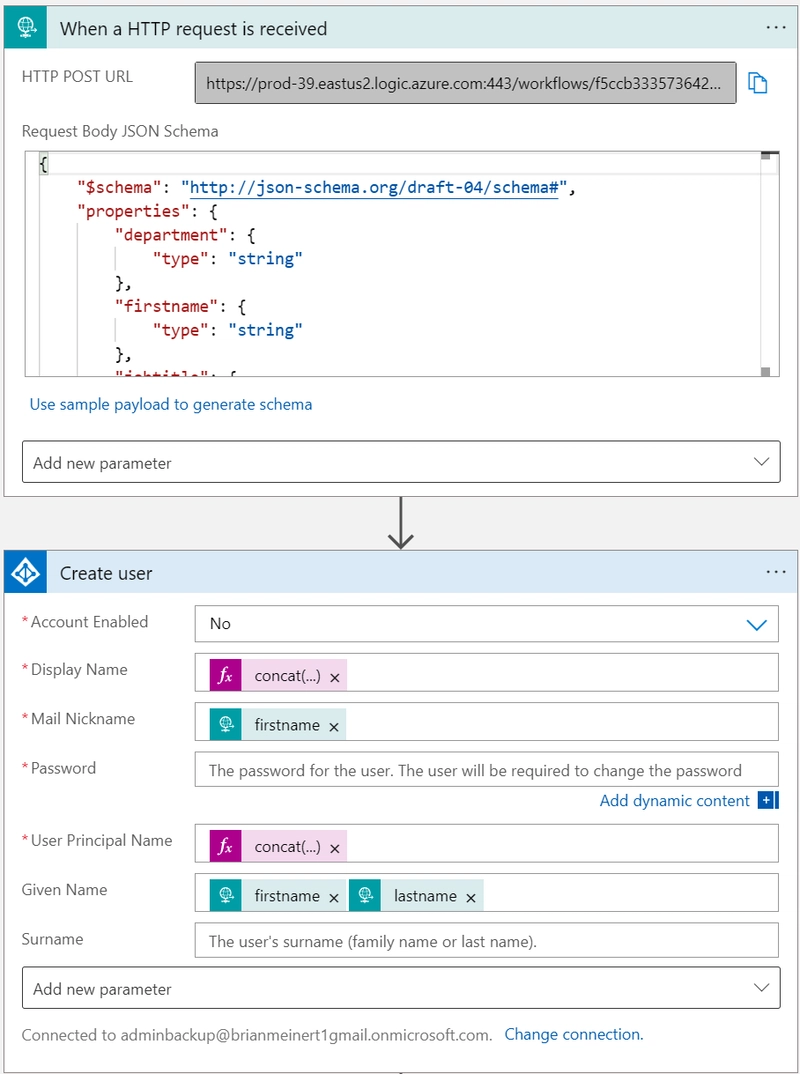


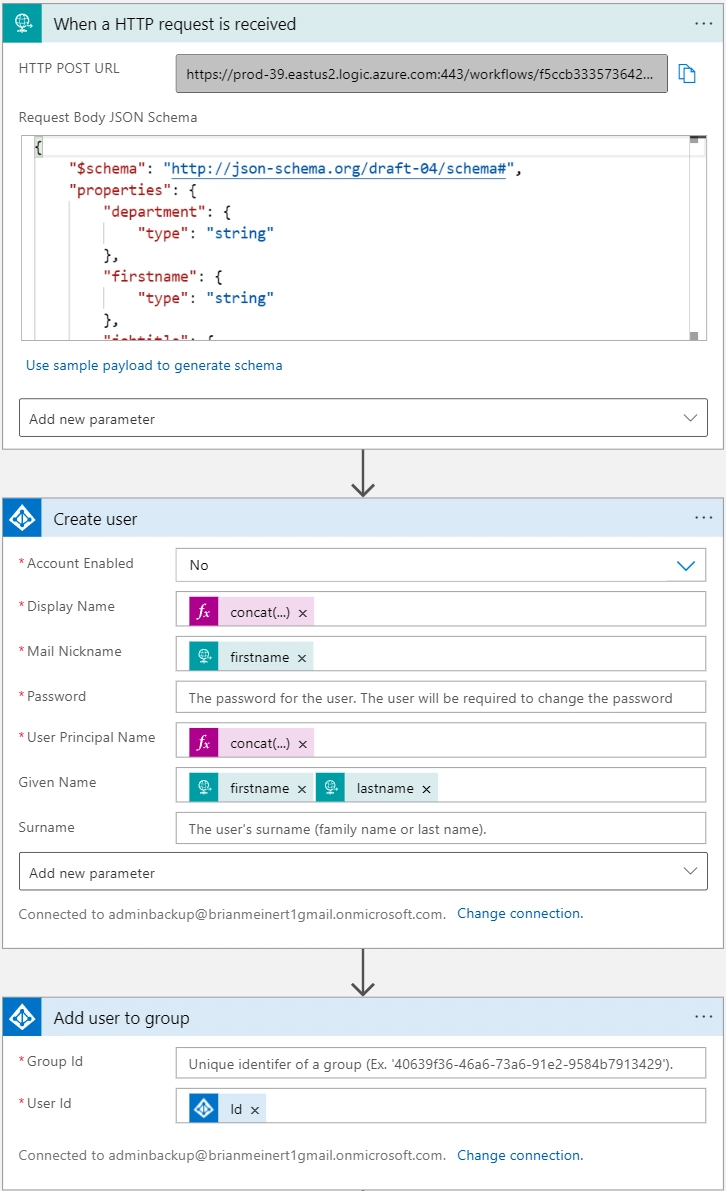
1. With the JSON data and generated passwords, user accounts are created in Azure Active Directory / Entra ID.

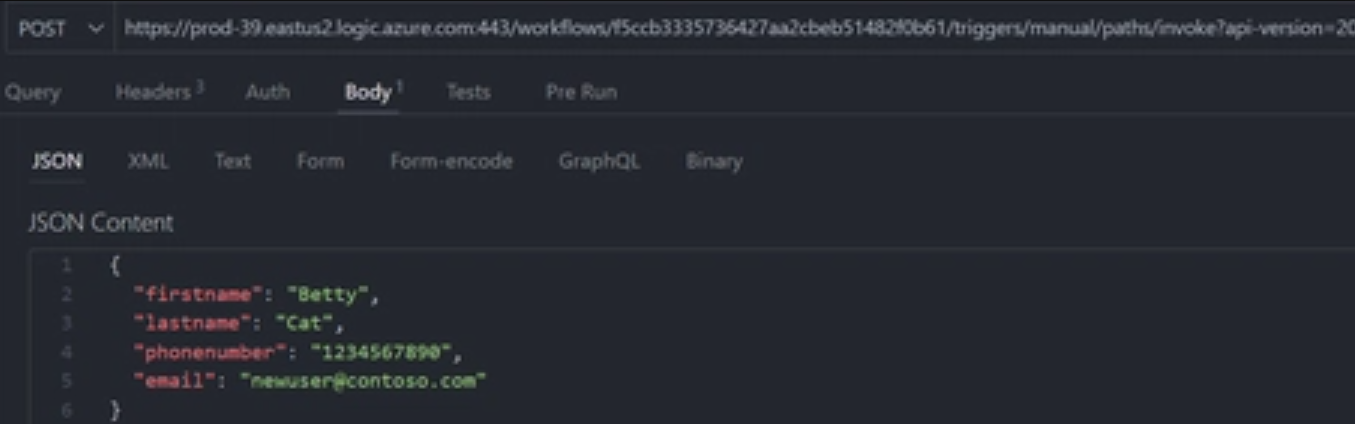


Review JSON file









**✅ Step 6: Email Temporary Credentials to Manager**

Once the user accounts are created in Entra ID (Azure AD), the Logic App continues the flow:

**📌 What You Need:**

* Email address of the hiring manager (should be present in CSV or pre-configured in a lookup table)
* Temporary passwords (generated by HTTP Function App)

**💡 Suggested Steps in Logic App:**

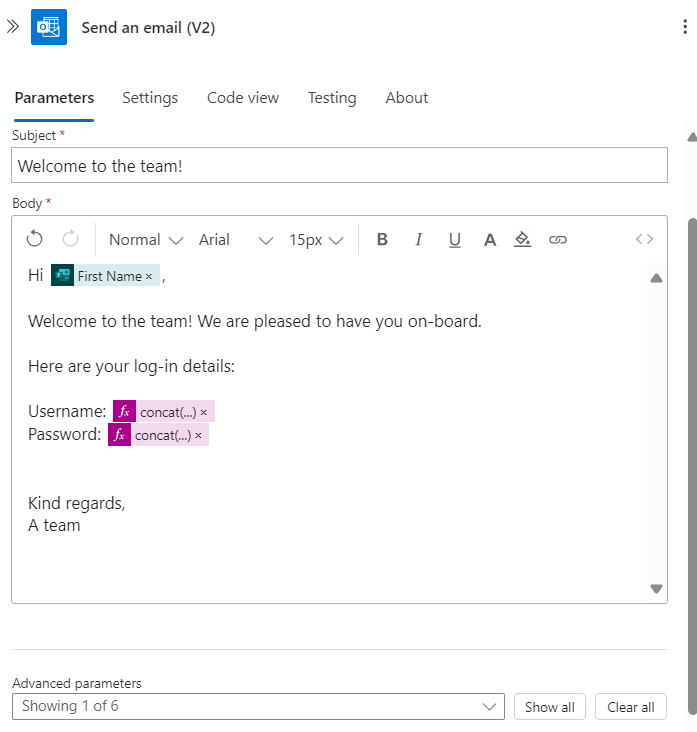
1. **Loop through each user in JSON**
2. **Create Entra ID user** using MS Graph API (or Azure AD connector)
3. Capture:
   * Username
   * Generated secure password
4. **Compose an email body** using Html Table or Create HTML table from array
5. Use **"Send Email (V2)"** from Office 365 Outlook or SMTP connector
   * To: Manager's email
   * Subject: "Temporary Credentials for New Joiners"
   * Body: Usernames + Temporary passwords

🧠 Tip: Store the manager’s email in a config table or metadata file if not in CSV.

**Automatically send a Welcome email:**

Lastly, automate sending a Welcome Email to the new user.

Use the Outlook (or SMTP) connector to send a personalised welcome message along with the login details.



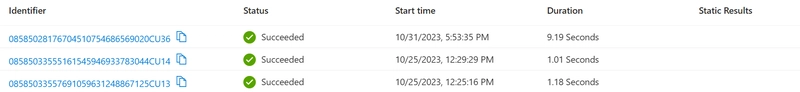
**✅ Step 7: Archive CSV and Notify HR**

**📌 What You Need:**

* Completed folder path in Blob Storage
* HR email address

**💡 Suggested Steps in Logic App (after all users are processed):**

1. **Move Blob File**
   * Use "Get Blob Content" (if not already done)
   * Use "Create Blob" in the /completed/ folder
   * Use "Delete Blob" in the original folder
2. **Send Email Notification to HR**
   * Use "Send Email (V2)" action
   * Subject: “Onboarding Completed”
   * Body: Summary – number of users processed, errors (if any), file name, timestamp

checked the Entra ID for the new user and also made sure the user was in the group, followed by checking my email to make sure that I got the email to verify and activate the account.  


Another way using:

Using **Microsoft forms connector**.

<https://www.linkedin.com/pulse/automate-entra-user-onboarding-azure-logic-apps-angel-ramirez-ymr0c/>

**Automate Onboarding New Users Into Entra ID**

<https://medium.com/@dhearn219/automate-onboarding-new-users-into-entra-id-a0c9d7808650>