Based on the transcript you provided, the hackathon demo showcases how to implement **customized MFA (Multi-Factor Authentication) enrollment and enforcement** using **Okta Customer Identity Cloud**. The speaker is demonstrating a local web app that uses Okta for authentication and MFA, with varying behaviors based on user attributes or group membership.

Here’s a breakdown of **what’s already done** and **what still needs to be implemented** for the hackathon:

**✅ Already Implemented / Demonstrated:**

1. **Custom Login Integration with Okta Sign-In Widget**
   * Local app with Okta sign-in widget (hosted or embedded)
   * Custom UI branding (logo, background, colors)
   * Username + password authentication
   * Google social login option (visible but not demoed)
2. **Three Demo Users with Different MFA Behaviors**
   * **Jack Smith**: No MFA required (mfa\_enforced = false)
   * **Mike Smith**: MFA is enforced, prompted to enroll if not already enrolled
   * **Tom Smith**: Can *opt into* MFA enrollment from within the app
3. **MFA Enrollment Flow**
   * Google Authenticator setup (QR code scan + OTP entry)
   * Once enrolled, login requires OTP code
4. **Custom Profile Attribute (mfa\_enforced)**
   * Added via Okta profile editor
   * Also added as a **custom claim** in the custom authorization server
   * Used to control UI rendering and backend logic
5. **Dynamic Group and Group Rule Configuration**
   * A dynamic group (Empower MFA TOTP Required) is created
   * A rule links user profile attribute (mfa\_enforced = true) to this group
   * Group membership enforces MFA at policy level

**🛠️ To Be Implemented for the Hackathon:**

| **Feature** | **Description** |
| --- | --- |
| **1. In-app MFA Opt-in Toggle** | UI toggle/button to allow users like Tom to **opt into MFA** from within the application |
| **2. API Call to Set Profile Attribute** | When user opts in, backend should make an **API call to Okta** to set mfa\_enforced = true |
| **3. Dynamic Group Rule Setup in Okta** | Define and test dynamic rule: user.mfa\_enforced == true → add to group enforcing MFA |
| **4. Token Handling** | ID token should contain mfa\_enforced claim. App must use this claim to determine whether to prompt or allow MFA enrollment |
| **5. Conditional MFA Enrollment Button** | Frontend logic: show or hide **“Enroll in MFA”** button based on mfa\_enforced claim in ID token |
| **6. MFA Status Check (Optional)** | Optional API call (to Okta or your backend) to determine whether the user has MFA enrolled already |
| **7. Group Assignment UI (Optional)** | (Optional for admin users) UI toggle that assigns user to the MFA group directly instead of using dynamic rules |
| **8. Social Login Testing (Optional)** | Google login is visible but not working in the demo. Optional to configure and test. |

**💡 Optional Enhancements (Stretch Goals):**

* **Audit Trail or Logs** for when a user opts in or is assigned MFA
* **Support for additional authenticators** (e.g., Okta Verify, Email OTP, etc.)
* **Error handling and fallback UX** if MFA enrollment fails
* **Revoke MFA enrollment** (button to opt out, if allowed)

**🔧 Tools/Skills Needed:**

* Okta Admin Console:
  + Profile editor
  + Authorization server (custom claims)
  + Group rules
* Okta API:
  + Update user profile attributes
  + Get user MFA status
* Frontend Logic:
  + Token parsing (ID token claims)
  + Conditional rendering of MFA UI
* Backend (Node.js / Spring Boot / etc.):
  + API calls to Okta
  + Token decoding / validation

Let me know if you want sample code for:

* Making the profile update API call
* Handling custom claims in the frontend
* Setting up group rules in Okta

Or if you're using React or another specific stack, I can help tailor the implementation accordingly.