Here's a general outline of the system architecture:

1. **User Authentication:**
   * Implement user authentication to ensure that only authorized users can access the application.
   * Use a secure authentication method such as username/password authentication or token-based authentication.
2. **Multi-Level Privileges:**
   * Define different user roles or categories, such as Admins, Authorities, Class Teachers students of five classes(or more).
   * Assign privileges to each role based on the level of access they should have.
   * Examples of privileges could include read-only access, read-write access, or access to specific models and data.
3. **Integrate Authentication into Streamlit(For the prototype only):**
   * Leverage Streamlit's session state to store user authentication information securely.
   * Create a login page where users can enter their credentials.
   * Verify user credentials against a secure database or authentication service(Supabase with Clerk).
4. **Authorization Middleware:**
   * Implement middleware to check user privileges before allowing access to certain features or data.
   * Use decorators or functions that wrap around specific parts of the application to enforce authorization.
5. **Secure API Keys and Credentials:**
   * Store sensitive information such as Google and other API keys and credentials securely.
   * Avoid hardcoding credentials directly in the application code.
   * Use environment variables or a secure configuration mechanism to manage sensitive information.
6. **Logging and Monitoring:**
   * Implement logging to keep track of user activities and potential security issues.
   * Set up monitoring to detect unusual or suspicious behavior.
7. **HTTPS and SSL/TLS:**
   * Ensure the application is served over HTTPS to encrypt data in transit.
   * Use SSL and TLS certificates to secure the communication between clients and the server.
8. **Regular Security Audits:**
   * Conduct regular security audits to identify and address potential vulnerabilities.
   * Keep all libraries and dependencies up to date to benefit from security patches.
9. **Data Encryption and Redundancy:**
   * Encrypt sensitive data stored in the application's database.
   * Use encryption algorithms to protect user credentials and other sensitive information.
   * Use multiple database from multiple vendor in multiple region to ensure database remains up at all time.
10. **Rate Limiting:**

* Implement rate limiting to prevent abuse or unauthorized access through brute-force attacks.