Technical architecture:

1. **Frontend (Client-side):**

   - Use a web-based frontend framework like React or Angular for a dynamic user interface.

   - Implement responsive design for accessibility on different devices.

   - Utilize a state management system (e.g., Redux for React) to manage application state.

2. **Backend (Server-side):**

   - Choose a backend framework like Node.js (Express), Django, or Flask.

   - Design a RESTful API to handle communication between the frontend and backend.

   - Implement user authentication and authorization for secure access to the application.

3. **Database:**

   - Use a relational database such as PostgreSQL or MySQL to store employee data, travel requests, and approvals.

   - Design a schema that efficiently represents the relationships between employees, travel requests, and approvals.

4. **Server Logic:**

   - Implement business logic for creating, updating, and retrieving travel requests.

   - Include workflows for approval processes, considering different levels of authorization.

   - Ensure data validation and error handling to maintain data integrity.

5. **Integration with External Systems:**

   - Integrate with corporate systems, such as HR systems for employee data and email systems for notifications.

   - Utilize APIs for currency conversion, weather updates, or any other external data needed for travel requests.

6. **Notifications:**

- Implement a notification system to alert employees and approvers of the status of their travel requests.

- Use email notifications or push notifications depending on the corporate preferences.

7. **Security:**

- Implement secure coding practices to prevent common web vulnerabilities.

- Use HTTPS for secure data transmission.

- Regularly update dependencies and perform security audits.

8. **Testing:**

- Conduct unit testing for individual components and integration testing for the entire system.

- Implement automated testing for regression testing and continuous integration.

9. **Deployment:**

- Deploy the application on a cloud platform like AWS, Azure, or Google Cloud for scalability and reliability.

- Utilize containerization (e.g., Docker) for easy deployment and scaling.

10. **Monitoring and Logging:**

- Implement logging mechanisms for tracking application behavior.

- Set up monitoring tools to detect and respond to issues promptly.

11. **Documentation:**

- Create comprehensive documentation for developers, administrators, and end-users.

- Include API documentation, database schema, and deployment instructions.