Slide 1: Intro Harsha

Slide 2: System Level Architecture Harsha

**Slide 3: Vanshaj**

**University Explorer Frontend Architecture: Technical Report**

**Overview**

The University Explorer frontend is built using React.js with a focus on component reusability, responsive design, and user experience. The application implements a modern single-page application (SPA) architecture with client-side routing and state management through React Context API.

**Technology Stack**

**Core Technologies**

* **React.js (v19.0.0)**: Modern UI library for building component-based interfaces
* **React Router (v7.4.0)**: Client-side routing framework
* **Axios**: Promise-based HTTP client for API requests
* **Reactstrap/Bootstrap 5**: UI component framework providing responsive design patterns
* **JWT-decode**: Utility for decoding JWT tokens in authentication flow

**Development Tools**

* **Create React App**: Project scaffolding and build configuration
* **React Scripts**: Development server and build tools
* **ESLint**: Code quality and style checking
* **Testing Library**: Component testing utilities

**Slide 4: Vanshaj**

**Project Architecture**

The client implements a well-structured organization with clear separation of concerns:

**1. Application Structure**

* **Entry Point (**[index.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**)**: Renders the root App component and applies the Bootstrap styling
* **App Component (**[App.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)**)**: Defines application routing and wraps the application in the authentication context
* **API Layer (api.js)**: Centralizes API communication with the backend

**2. Component Organization**

The application follows a hierarchical component structure:

* **Layout Components**:
  + [components/Navbar.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Responsive navigation bar with dynamic menu based on authentication status
* **Authentication Components**:
  + [components/ProtectedRoute.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Route wrapper that enforces authentication
* **Page Components**:
  + **Public Pages**:
    - [pages/Home.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Landing page with application overview
    - [pages/Login.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): User authentication form
    - [pages/Register.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): User registration form
  + **Authenticated Pages**:
    - [pages/Universities.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): University listing with filtering and sorting
    - [pages/UniversityDetails.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Detailed university information
    - [pages/Profile.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): User profile information
    - [pages/Favorites.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): User's favorite universities
  + **Admin Section**:
    - [pages/admin/AdminDashboard.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Overview of system entities
    - [pages/admin/UniversityList.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): University management
    - [pages/admin/ProgramList.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Program management
    - [pages/admin/UserList.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): User management
    - [pages/admin/AddUniversity.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): University creation form
    - [pages/admin/AddProgram.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): Program creation form
    - [pages/admin/AddUser.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html): User creation form

**3. State Management**

The application uses React's Context API for global state management:

* **Authentication Context** ([context/AuthContext.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)):
  + Manages user authentication state
  + Provides login/logout functionality
  + Handles JWT token storage and retrieval
  + Exposes user data to components

**4. Routing System**

The application uses React Router with a sophisticated routing structure:

* Public routes accessible to all users
* Protected routes requiring authentication
* Role-based routes for administrative access
* Nested routes for related content

<AuthProvider>

  <Router>

    <Navbar />

    <Routes>

      <Route path="/" element={<Home />} />

      <Route path="/login" element={<Login />} />

      <Route path="/register" element={<Register />} />

      <Route element={<ProtectedRoute />}>

        <Route path="/universities" element={<Universities />} />

        <Route path="/universities/:id" element={<UniversityDetails />} />

        <Route path="/favorites" element={<Favorites />} />

        <Route path="/profile" element={<Profile />} />

        <Route path="/admin" element={<AdminDashboard />} />

        <Route path="/admin/universities" element={<UniversityList />} />

        <Route path="/admin/programs" element={<ProgramList />} />

        <Route path="/admin/users" element={<UserList />} />

        <Route path="/admin/universities/add" element={<AddUniversity />} />

        <Route path="/admin/programs/add" element={<AddProgram />} />

        <Route path="/admin/users/add" element={<AddUser />} />

      </Route>

    </Routes>

  </Router>

</AuthProvider>

**Slide 5: Vanshaj**

**Key Features & Implementation Details**

**1. Authentication System**

The frontend implements a comprehensive JWT-based authentication system:

* **Token Storage**: Persistent storage in [localStorage](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ")
* **Automatic Token Handling**: Axios interceptors attach tokens to API requests
* **Session Management**: Token retrieval on page refresh
* **Protected Routes**: Authentication state verification before rendering private routes

**2. University Exploration**

The [pages/Universities.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) component offers rich exploration features:

* **Searching**: By university name
* **Filtering**: By location and program offerings
* **Sorting**: By ranking or name, with ascending/descending options
* **Favoriting**: Adding universities to a user's favorites list

**3. Admin Dashboard**

The admin section provides comprehensive management capabilities:

* **Dashboard Overview**: [pages/admin/AdminDashboard.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) with entity stats
* **Entity Management**: CRUD operations for universities, programs, and users
* **Form Validation**: Input validation for creating and editing entities

**4. User Interface Design**

The frontend implements a modern, responsive design using Bootstrap/Reactstrap:

* **Responsive Grid System**: Content adapts to different screen sizes
* **Card-Based UI**: Content is presented in clear, separated card components
* **Consistent Navigation**: Navbar adapts based on authentication state
* **Visual Feedback**: Loading spinners, alerts for success/error states
* **Iconography**: React Icons provide visual context to actions and data

**5. Data Fetching & Error Handling**

The application implements robust data fetching patterns:

* **Loading States**: Visual indicators during API requests
* **Error Handling**: User-friendly error messages
* **Empty States**: Appropriate messaging when no data is available
* **Debounced Search**: Performance optimization for search inputs

**Component Communication Patterns**

The frontend uses several patterns for component communication:

1. **Parent-Child Props**: For direct data passing
2. **Context API**: For global state like authentication
3. **URL Parameters**: For entity identification (e.g., university ID)
4. **Custom Hooks**: For reusable state logic

**Performance Optimization**

The application implements several performance optimizations:

* **Lazy Loading**: Route-based code splitting (implicit in React Router)
* **Debounced Search**: Preventing excessive API calls during typing
* **Conditional Rendering**: Components render only when data is available
* **Memoization**: useEffect dependencies prevent unnecessary re-renders

**Security Implementations**

The frontend incorporates several security practices:

* **Protected Routes**: Authentication check before rendering private content
* **Token-Based Authentication**: Secure JWT handling
* **Role-Based Access Control**: Admin-only routes and features
* **Form Validation**: Input sanitization before submission
* **HTTPS**: Assumed secure communication with backend

**User Experience Considerations**

The application prioritizes user experience through:

* **Responsive Design**: Adapts to different device sizes
* **Loading States**: Visual feedback during asynchronous operations
* **Error Handling**: User-friendly error messages
* **Intuitive Navigation**: Logical information architecture
* **Visual Hierarchy**: Clear content prioritization
* **Consistent Styling**: Unified color scheme and component design

**Slide 6: Harsha**

**Overview**

The University Explorer application's backend is built using a Node.js and Express.js framework with MongoDB as its database. The server implements a comprehensive RESTful API architecture that follows modern web development patterns and security practices.

**Technology Stack**

**Core Technologies**

* **Node.js**: JavaScript runtime environment for server-side execution
* **Express.js**: Web application framework for handling HTTP requests, routing, and middleware
* **MongoDB**: NoSQL database for data storage
* **Mongoose**: Object Data Modeling (ODM) library for MongoDB that provides schema validation and relationship modeling

**Authentication & Security**

* **JWT (JSON Web Tokens)**: Used for authentication and maintaining user sessions
* **bcryptjs**: Password hashing library for secure storage of user credentials
* **helmet**: Security middleware that sets various HTTP headers to enhance security
* **express-rate-limit**: Middleware to limit repeated requests to public APIs
* **express-validator**: Middleware for request data validation
* **express-mongo-sanitize**: Middleware to prevent NoSQL injection attacks

**Slide 7: Harsha**

**Project Architecture**

The server follows a modular architecture with clear separation of concerns:

**1. Entry Point (server.js)**

The main application file configures Express, connects to MongoDB, sets up middleware, and defines routes:

* Sets up security headers with [helmet](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)
* Implements request rate limiting
* Registers API routes
* Configures CORS for cross-origin requests
* Connects to MongoDB via Mongoose

**2. Models (Mongoose Schemas)**

Models define the structure and validation rules for database collections:

* **University Model** (models/University.js): Contains university details including:
  + Name, location, ranking, description, website
  + Relationship with programs via references
* **Program Model** (models/Program.js): Represents academic programs with:
  + Name, duration, degree type (e.g., BSc, MSc, PhD)
  + Relationship to universities
* **User Model** (models/User.js):
  + Authentication details (email, password)
  + Role-based access control (admin/user roles)
  + Password hashing functionality
  + Support for saving favorite universities
* **Review Model** (models/Review.js):
  + Ratings and comments for universities
  + Relationships to both universities and users

**3. Controllers**

Controllers implement business logic for each route:

* **University Controller** (controllers/universityController.js):
  + CRUD operations for universities
  + Advanced filtering and sorting logic
  + Pagination implementation
  + MongoDB aggregation pipeline for complex queries
* **Program Controller** (controllers/programController.js):
  + CRUD operations for programs
  + Manages program-university relationships
* **Auth Controller** (controllers/authController.js):
  + User registration with validation
  + Login with password comparison
  + JWT token generation
* **Admin Controller** (controllers/adminController.js):
  + User management functions
  + Role modification capabilities
  + Admin-only operations

**4. Routes**

Routes define API endpoints and connect them to controllers:

* **Auth Routes** (routes/auth.js): Authentication endpoints with validation
* **University Routes** (routes/universities.js): University CRUD operations with protection
* **Program Routes** (routes/programs.js): Program management endpoints
* **User Routes** (routes/user.js): User-specific functions including favorites
* **Review Routes** (routes/reviews.js): Review creation and retrieval
* **Admin Routes** (routes/admin.js): Administrator-specific functionality

**5. Middleware**

Custom middleware functions provide cross-cutting concerns:

* **Authentication Middleware** (middleware/authentication.js):
  + JWT verification
  + User identification from tokens
  + Role-based access control via [adminMiddleware](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html" \o ")

**6. Database Connection**

Database connection is managed in [connect.js](vscode-file://vscode-app/Applications/Visual%20Studio%20Code.app/Contents/Resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) using Mongoose to establish and maintain the MongoDB connection.

**7. Data Seeding**

The application includes a sophisticated data seeding system (db/seed.js) that:

* Populates the database with sample universities, programs, users, and reviews
* Creates relationships between entities
* Clears existing data before seeding
* Implements random generation of related entities

**Slide 8: Prashast**

**API Endpoints**

The API structure follows RESTful conventions:

**Authentication APIs**

* POST /api/auth/register: User registration
* POST /api/auth/login: User authentication
* GET /api/auth/me: Get current user profile

**University APIs**

* GET /api/universities: List universities with filtering, sorting, and pagination
* GET /api/universities/:id: Get university details with programs
* POST /api/universities: Create university (admin only)
* PUT /api/universities/:id: Update university (admin only)
* DELETE /api/universities/:id: Delete university (admin only)

**Program APIs**

* GET /api/programs: List programs with filtering and sorting
* GET /api/programs/:id: Get program details
* POST /api/programs: Create program (admin only)
* PUT /api/programs/:id: Update program (admin only)
* DELETE /api/programs/:id: Delete program (admin only)

**User APIs**

* GET /api/user/profile: Get user profile
* GET /api/user/favorites: Get user's favorite universities
* POST /api/user/favorites/:universityId: Add university to favorites
* DELETE /api/user/favorites/:universityId: Remove university from favorites

**Review APIs**

* GET /api/reviews/:universityId: Get reviews for a university
* POST /api/reviews/:universityId: Add a review for a university
* DELETE /api/reviews/:reviewId: Delete a review

**Admin APIs**

* GET /api/admin/users: List all users (admin only)
* PUT /api/admin/users/:id: Update user role (admin only)
* DELETE /api/admin/users/:id: Delete user (admin only)

**Slide 9: Prashast**

**Security Features**

The backend implements several security practices:

1. **Authentication**:
   * JWT-based authentication with token expiration
   * HTTP-only cookies for token storage
   * Password hashing with bcrypt and salting
2. **Authorization**:
   * Role-based access control (admin/user)
   * Route protection with middleware
   * Resource ownership validation for personal data
3. **API Security**:
   * Rate limiting to prevent brute force attacks
   * Security headers with Helmet.js
   * CORS configuration for frontend access
   * Input validation with express-validator

**Database Design**

The MongoDB database uses document relationships with:

1. **References**: Using MongoDB ObjectIds to create relationships between documents
2. **Schema Validation**: Using Mongoose to enforce data structure rules
3. **Indexing**: Implied from query patterns in controllers

**Slide 10: Prashast**

**Future Expansion Areas**

The frontend architecture is designed for scalability with potential areas for expansion:

1. **Advanced Search**: More sophisticated filtering and comparison tools
2. **User Settings**: Personalization options and preferences
3. **Analytics Dashboard**: Data visualization for administration
4. **Internationalization**: Multi-language support
5. **Accessibility Enhancements**: ARIA attributes and keyboard navigation