1. Server Setup (api/server.ts)

- **Frameworks & Libraries:**

Uses **Express** for HTTP API, **Socket.IO** for real-time communication, **Multer** for file uploads, and **Bull/Redis** for job queues (if Redis is available).

- **CORS & Middleware:**

Configured to allow requests from local frontends. Handles large JSON and URL-encoded payloads.

- **File Uploads:**

Uses Multer to accept only CSV files, storing them in an uploads/ directory.

- **WebSocket (Socket.IO):**

Tracks active connections, allows clients to join "campaign rooms" for real-time updates (e.g., broadcast progress).

2. API Endpoints

Health Check

- GET /api/health

Returns status of API, WebSocket, and WhatsApp services.

CSV Handling

- POST /api/analyze-csv
- Accepts a CSV file upload.
- Parses the CSV, returns available columns, a sample of the data, and a temporary file path for further processing.
- Suggests expected fields (name, email, phone, etc.).
- POST /api/process-csv
- Accepts a temp file path and a field mapping (mapping CSV columns to expected fields).
- Transforms and validates the data using Zod schemas.
- Filters out invalid or empty rows, generates placeholder emails if needed.
- Saves the processed data for later use (e.g., for broadcasting).
- Returns stats and breakdowns (roles, years, branches).

3. Contacts API (api/routes/contacts.ts)

- **Mirrors the CSV endpoints above** (analyze, process) for modularity.
- Maintains in-memory storage of the current contacts data.
- Exports helpers to get/set the current contacts, used by other modules (like WhatsApp broadcasting).

4. WhatsApp API (api/routes/whatsapp.ts)

- **Status & QR Code:**
- GET /status: Returns WhatsApp client connection/authentication status.
- GET /qr: Returns a QR code for WhatsApp authentication (placeholder in demo).
- **Broadcasting:**
- POST /broadcast: Starts a new broadcast campaign.
- Accepts a message, campaign name, and optional filters.
- Filters contacts as needed.
- Creates a campaign object and simulates sending messages (replace with real WhatsApp integration in production).
- Tracks progress (sent, failed, total).
- Notifies clients via WebSocket.
- **Campaign Management:**
- GET /campaigns: Lists all campaigns.
- GET /campaigns/:id: Gets details of a specific campaign.

5. Message Preview API (api/routes/preview.ts)

- POST /api/preview
- Accepts a message template and a contact object.
- Substitutes {{variable}} placeholders in the template with contact data.
- Returns the preview and a list of missing variables.
  -

6. Utilities & Validation

- **Validation:**

Uses Zod schemas (from script/validation.ts)
to ensure data integrity for contacts and broadcasts.

- **Helpers:**

Utility functions for CSV/JSON conversion, breakdowns by field, and filtering.

7. Data Flow Example

1. **Upload CSV:**

User uploads a CSV of contacts via /api/analyze-csv.

2. **Map Fields:**

User maps CSV columns to required fields via /api/process-csv.

3. **Preview & Filter:**

User previews messages and filters contacts as needed.

4. **Broadcast:**

User starts a broadcast via /api/whatsapp/broadcast.Backend sends messages (simulated or real), tracks progress, and updates the frontend in real time.

5. **Campaign Tracking:**

User can view campaign status and results via /api/whatsapp/campaigns.

8. Real-Time Updates

- **WebSocket:**

Used for real-time campaign progress updates.
Clients join rooms for specific campaigns to receive updates as messages are sent.