**Part 3 - Detailed Design**Created By: Ido Bashari & Lior Toledano

**Architecture**

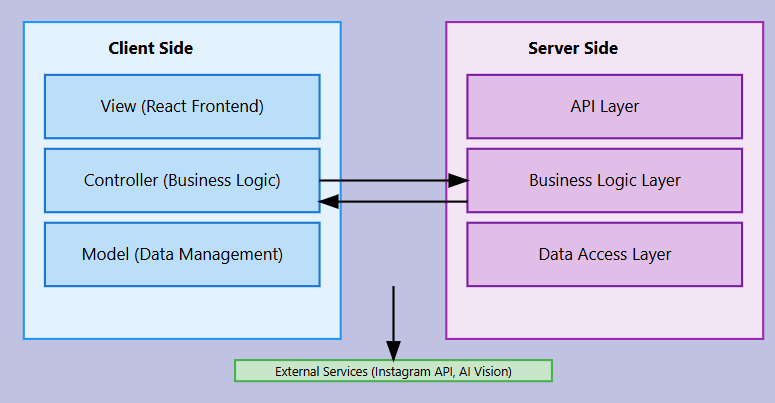
The architecture that best suits our project is **Client-Server with MVC Pattern**.

**Client-Server with MVC Architecture in our project:**

* **Model**:
  + Manages all data operations including user profiles, artist portfolios, and search history
  + Handles data retrieval and storage through Supabase
  + Processes AI image analysis results and style matching
* **View**:
  + React-based web interface
  + Presents data to users and artists
  + Handles user input and interactions
  + Displays search results and artist portfolios
* **Controller**:
  + Acts as a bridge between Model and View
  + Processes user requests and updates the Model accordingly
  + Manages business logic for search operations and artist matching
  + Handles integration with external services (Instagram API, AI Vision Services)

**Data Storage:**

* The system data is stored in Supabase (PostgreSQL)
* Artist portfolios and images are synchronized from Instagram
* ML models and AI analysis results are cached in Redis

**Graphic Description:**  
  
**data description:**

● **Users list** - a database table that has the following fields:  
 ○ user ID   
○ username  
 ○ email   
○ password hash   
○ profile type (artist/client)   
○ registration date   
○ status (active/inactive)   
○ preferences settings

● **Artists list** - a database table that has the following fields:   
○ artist ID   
○ user ID reference   
○ instagram handle   
○ biography text   
○ location coordinates   
○ service area  
 ○ rating average   
○ verified status   
○ last portfolio sync date

● **Styles list** - a database table that has the following fields:   
○ style ID   
○ style name   
○ main category   
○ subcategory   
○ description   
○ characteristic attributes   
○ popularity score

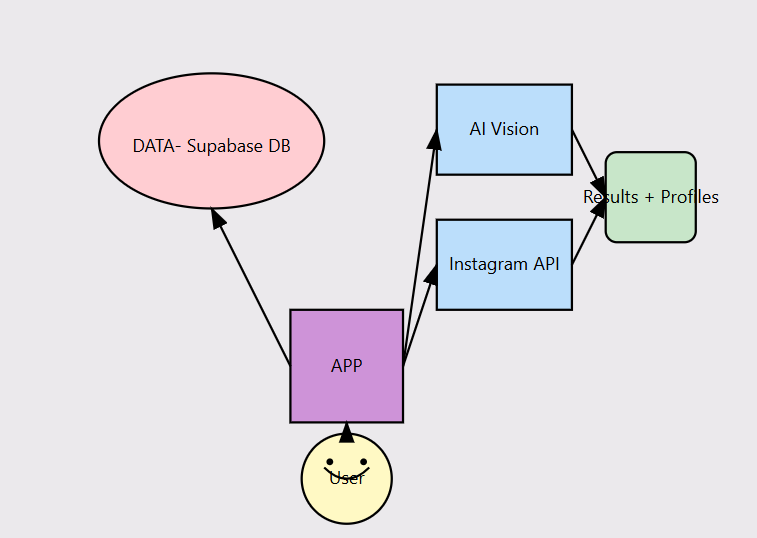
● **Reviews list** - a database table that has the following fields:   
○ review ID   
○ artist ID reference   
○ client ID reference   
○ rating score   
○ review text   
○ date of service   
○ review date   
○ verification status

**API specification:**

User interface:

* Register/Login to system (as artist or client)
* Connect Instagram account for artists
* Upload reference image for AI style analysis
* Search artists by style/location
* Filter search results by:
  + Style categories
  + Location/distance
  + Rating
  + Verification status
* View artist portfolios and Instagram feeds
* Save favorite artists
* Write and view reviews
* Message artists through the platform
* Update profile settings
* View search history
* Save and share artist profiles
* Download portfolio images

**interface design:**

  
  
  
  
  
**Programming Languages and Tools** -

The reason we selected HTML, CSS and JavaScript is that we need to create a web-based application that is accessible and responsive. For that reason, we chose to work with:

* HTML for structure
* CSS for styling and responsive design
* JavaScript for client-side interactions
* Node.js for server-side logic

Also, we will use React library for creating user interfaces and Supabase for database management.

**Algorithm Description**

The INKSCAPE system relies on three core algorithmic components that work together to provide accurate tattoo artist matching:

1. **Image Analysis and Style Recognition Algorithm**
   * **Purpose**: Analyzes uploaded reference images to identify tattoo styles, elements, and characteristics
   * **Input**: User-uploaded image
   * **Output**: Style classification, feature set, and element identification
   * **Process Flow**:
     1. Image Preprocessing
        + Format validation and standardization
        + Resolution adjustment
        + Color space normalization
     2. Feature Extraction (using Google Cloud Vision/Azure)
        + Edge detection and pattern recognition
        + Color palette analysis
        + Texture identification
     3. Style Classification
        + Comparison with pre-defined style templates
        + Multi-label classification for mixed styles
        + Confidence score calculation
   * **Time Complexity**: O(n\*m) where n is pixel count and m is feature set size

**Artist Matching Algorithm**

* **Purpose**: Matches users with tattoo artists based on style preferences, location, and other criteria
* **Input**:
  + Style classification results from image analysis
  + User location
  + Additional filters (price range, ratings, etc.)
* **Output**: Ranked list of matched artists with similarity scores
* **Process Flow**:
  + Style Matching
    - Compare identified styles with artist specializations
    - Calculate style compatibility score
    - Weight multiple style matches
  + Portfolio Analysis
    - Analyze artist's Instagram portfolio
    - Extract common elements and patterns
    - Compare with user preferences
  + Geographic Filtering
    - Calculate distances from user location
    - Apply service area restrictions
    - Weight results by proximity
  + Ranking Calculation
    - Combine all scoring factors:
      * Style match score (40%)
      * Portfolio similarity (30%)
      * Geographic proximity (20%)
      * Artist rating (10%)
* **Time Complexity**: O(k log k) where k is the number of active artists in the database

**Search and Filtering Algorithm**

* + **Purpose**: Provides real-time search capabilities with multiple parameters and dynamic filtering
  + **Input**:
    - Search parameters (styles, location, price range)
    - Filter selections
    - Sort preferences
  + **Output**: Filtered and sorted list of artists matching search criteria
  + **Process Flow**:
    - Query Processing
      * Parameter validation and normalization
      * Query optimization
      * Cache checking for similar recent queries
    - Multi-Parameter Filtering
      * Style-based filtering
      * Location-based filtering
      * Price range filtering
      * Rating threshold filtering
    - Result Aggregation
      * Merge results from different parameters
      * Remove duplicates
      * Apply secondary filters
    - Real-time Sorting
      * Dynamic result ordering based on:
        + Relevance score
        + Distance
        + Rating
        + Recent activity
    - Response Optimization
      * Pagination implementation
      * Cache result set
      * Prepare response format
  + **Time Complexity**: O(n log n) for sorting operations, where n is the number of filtered results
  + **Space Complexity**: O(n) for result set storage

**Implementation Notes**:

* Uses Redis for caching frequent queries
* Implements lazy loading for optimal performance
* Maintains in-memory indexes for common search parameters
* Employs geospatial indexing for location-based queries