Autimate API Documentation

What `AutiMate` does?

Our app are ment to be used by parents. A parent would answer few questions about child's behavior and upload a single 15-20 sec video clip of the child doing any activity (walking, grabbing objects, arm flipping etc.) and our system would check if child's behavior seemed to be autistic or not.
Our behavioral analysis model is a video classifier follows the architecture mentioned in this paper:

> *Rani, Asha and Verma, Yashaswi* (WACV 2024). **Activity-Based Early Autism Diagnosis Using a Multi-Dataset Supervised Contrastive Learning Approach**. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2024, pp. 7788-7797.

Technical Details

Our system consists of 4 different micro-services:

- Backend (Java Spring)
- Behavioral Analysis function - **Serverless** (Python, PyTorch, ONNX)
- Frontend (VueJS)
- Database (Cloud Mongodb)

Behavioral analysis function is a serverless function. The decision influenced by primarily for scaling factor and then cost factor. Because we haven't made the app public yet, we currently don't have a heavy load and serverless is therefore cost effective than renting a dedicated GPU cluster. At the same time, when we will go public, instead of vertically scaling, we can rely on serverless to handle multiple requeset at the same time.

How our serverless reduces load from backend?

Video classification requires immense resources. Doing it on backend server each time would left the 1 cpu core freeze from serving other requests for the time being. But this is unexpected and cannot be scaled. Therefore, we decided to move our video classification system on

serverless system. Here is a complete overview of how it works.

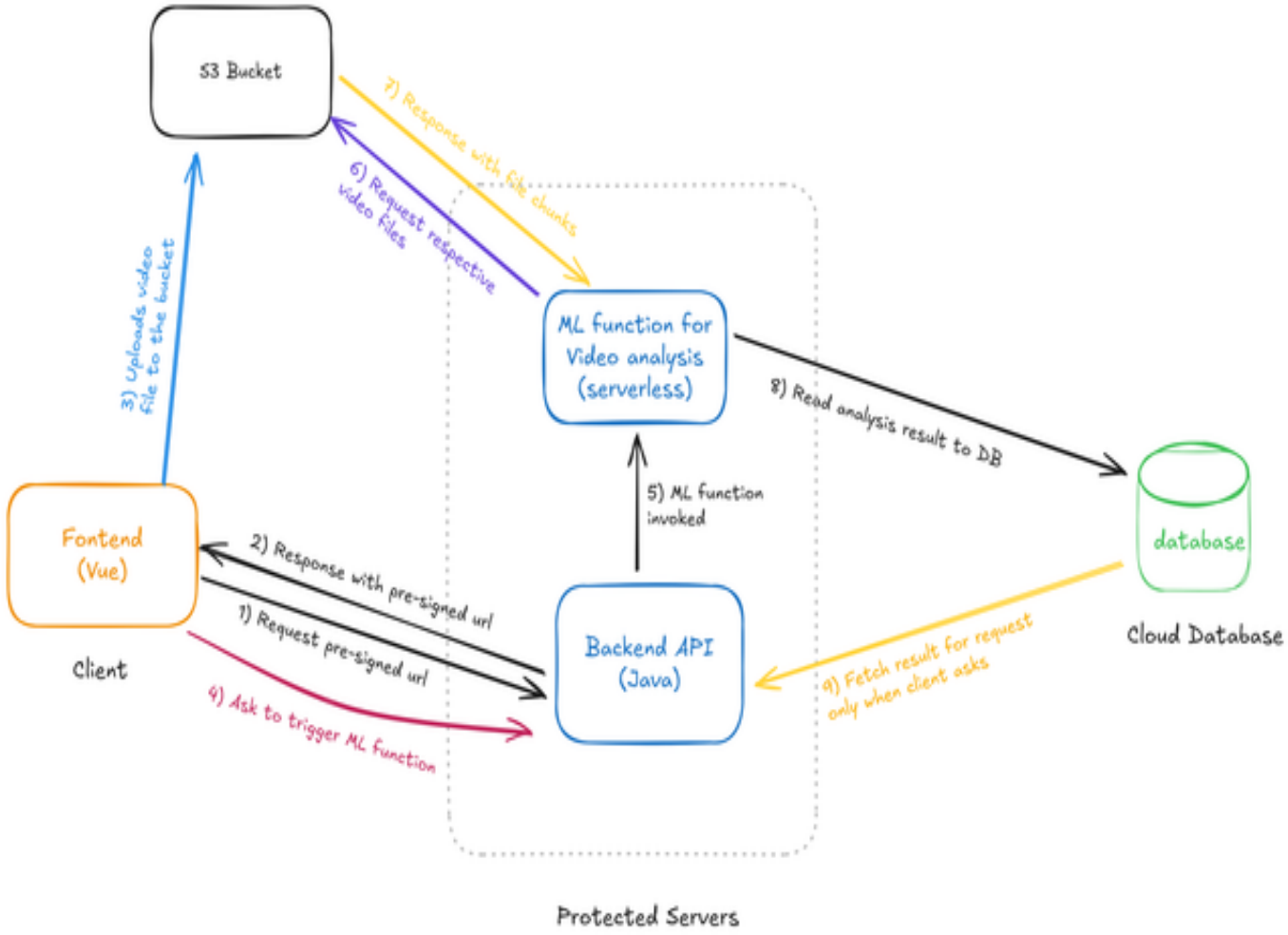

Fig 1: Video Classification system

How does feedback system works in therapy?

We tried to make it as real-time as possible. Unfortunately, we cannot afford a faster model (such as claude 3.5 haiku) at that moment and had to go with `gemini-1.5-flash` as it is the free option avilable. But, we believe that kids at our targeted age are quite slow at drawing and it feels kind of real time at their speed.

The overall procedure is quite simple. We start by randomly choosing a common object's picture. Then ask the child to draw the object.

Fig 2: Sample easy to draw image

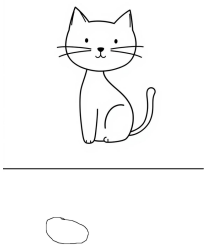

Fig 3: Reference image and drawn image combined to ask feedback from AI

The child's task is to draw something as close to the reference image. When any new stroke or line or shape is added to the canvas, we pair it along with the reference image and send to the backend server to request feedback from AI. The feedback then converted to sound via `SpeechSynthesisUtterance` API.

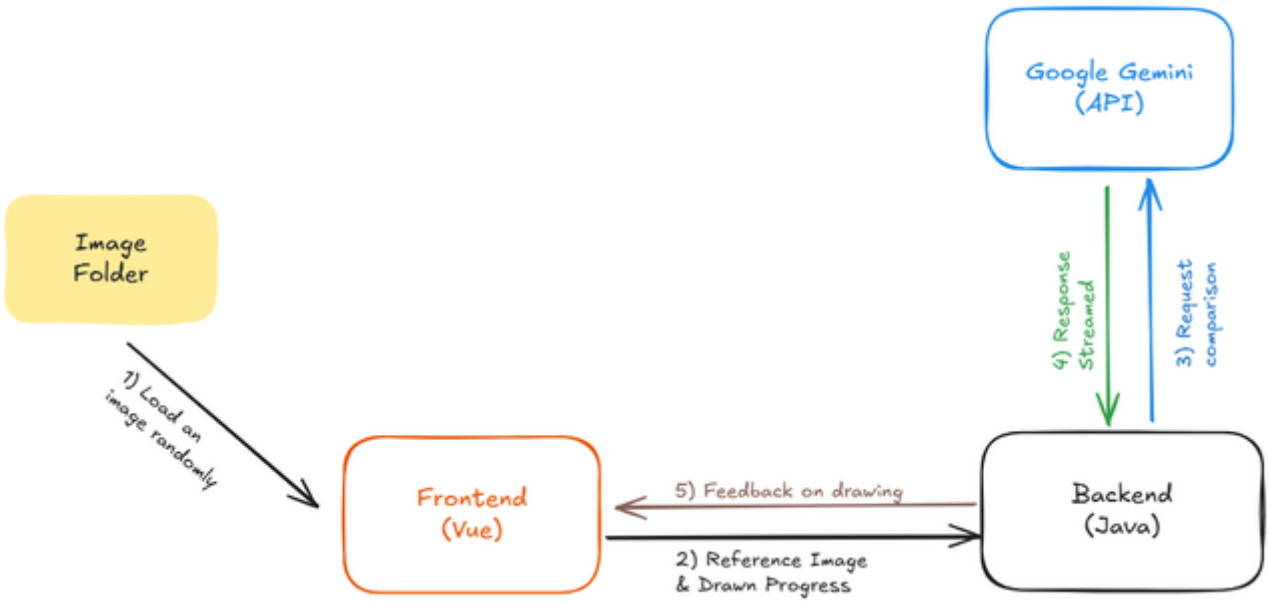

Fig 4: Drawing feedback system

Login test credentials

Admin

```
01902207410
holymoly
```

Parent/User

```
01902207411
password1
```

Explaination of API Endpoints and Implementation Details (Backend)

1. User Authentication APIs

1.1 Login

Endpoint: `POST /api/v1/auth/login`
Description: Authenticates a user using their phone number and password.

Request Headers:

- Authorization (Optional)

Request Body:

```
{
  "phone": "01902207412",
  "password": "holymoly"
}
```

Response: JWT token on successful authentication or error message.

Backend Implementation:

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody AuthRequest authRequest) {
    boolean isAuthenticated = authService.authenticate(authRequest.getPhone(),
authRequest.getPassword());
    if (isAuthenticated) {
        String token = jwtService.generateToken(authRequest.getPhone());
        return ResponseEntity.ok(Map.of("token", token));
    } else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid
credentials");
    }
}
```

Functional Details:

- `authService.authenticate()` : Validates the user's phone and password
- `jwtService.generateToken()` : Generates a JWT token for the authenticated user

1.2 Create New User

Endpoint: `POST /api/v1/auth/register`
Description: Registers a new user.

Request Body:

```
{
  "fullName": "An Admin",
  "phone": "011",
  "password": "mradmin"
}
```

Backend Implementation:

```
@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody UserRequest userRequest) {
    boolean isRegistered = userService.register(userRequest);
    if (isRegistered) {
        return ResponseEntity.ok("User registered successfully");
    } else {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Registration
failed");
    }
}
```

Functional Details:

- `userService.register()` : Handles user registration, including saving user details to the database

1.3 Request Verification OTP

Endpoint: `GET /api/v1/auth/send/otp/{phone}`
Description: Sends a verification OTP to the provided phone number.

Backend Implementation:

```
@GetMapping("/send/otp/{phone}")
public ResponseEntity<?> sendOtp(@PathVariable String phone) {
    boolean isSent = otpService.sendOtp(phone);
    return isSent ? ResponseEntity.ok("OTP sent") :
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to send OTP");
}
```

Functional Details:

- `otpService.sendOtp()` : Generates and sends an OTP to the given phone number

2. Admin APIs

2.1 Admin Login

Endpoint: `POST /api/v1/admin/auth/login`
Description: Logs in an admin user.

Backend Implementation:

```
@PostMapping("/admin/auth/login")
public ResponseEntity<?> adminLogin(@RequestBody AuthRequest authRequest) {
    boolean isAuthenticated = adminService.authenticate(authRequest.getPhone(),
authRequest.getPassword());
    if (isAuthenticated) {
        String token = jwtService.generateToken(authRequest.getPhone());
        return ResponseEntity.ok(Map.of("token", token));
    } else {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid
credentials");
    }
}
```

Functional Details:

- `adminService.authenticate()` : Validates admin credentials

3. Child Profile APIs

3.1 Add New Child Profile

Endpoint: `POST /api/v1/child/add`
Description: Adds a new child profile.

Request Headers:

- Authorization (Required)

Request Body:

```
{
  "name": "Child Name",
  "dob": "YYYY-MM-DD",
  "gender": "female"
}
```

Backend Implementation:

```
@PostMapping("/add")
public ResponseEntity<AuthenticationResponse> addChild(
        @RequestHeader("Authorization") String token,
        @RequestBody ChildRequest childRequest) {
    AuthenticationResponse response = childService.addChild(token,
childRequest.getName(), childRequest.getDob());
    return ResponseEntity.ok(response);
}
```

Functional Details:

- `childService.addChild()` : Adds the child profile, linking it to the authenticated user

4. Autism Evaluation APIs

4.1 Invoke Serverless Analysis

Endpoint: `POST /api/v1/aex/invoke`
Description: Invokes a serverless function for autism video analysis.

Request Body:

```
{
  "arrq10": [1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 7, 4, 1],
  "video_name": "1fcce48f62d044f2b669.mp4"
}
```

`video_name` is decided during requese and it is unique for each request.

Backend Implementation:

```
@PostMapping("/invoke")
public ResponseEntity<?> invokeServerless(@RequestBody AsdExRequest asdExRequest,
@RequestHeader("Authorization") String jwtToken) {
    try {
        String childId = childService.getActiveChild(jwtToken);
        asdExRequest.setChildId(childId);
        asdExRequest.setSecret_token(serverlessSecretToken);

        String serverlessUrl = "https://hossen1907012--autism-video-analysis-fn-
main.modal.run";
        String err = asdExServerlessInvokeService.invokeServerless(serverlessUrl,
asdExRequest);

        if (err == null) {
            return ResponseEntity.ok("Invocation successful");
        } else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Invocation
failed");
        }
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Internal
server error");
    }
}
```

Functional Details:

- `childService.getActiveChild()` : Retrieves the active child profile for the user
- `asdExServerlessInvokeService.invokeServerless()` : Invokes a serverless function with the provided request data

4.2 Therapy Suggestion

Endpoint: `POST /api/v1/aex/therapy_suggestion`
Description: Provides therapy suggestions based on user input.

Backend Implementation:

```
@PostMapping("/therapy_suggestion")
public ResponseEntity<?> getTherapySuggestion(@RequestBody
TherapySuggestionPromptStringRequest suggestionPromptStr,
@RequestHeader("Authorization") String jwtToken) {
    try {
        String childId = childService.getActiveChild(jwtToken);
        suggestionPromptStr.setChildId(childId);
        return ResponseEntity.ok("Therapy suggestion provided");
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Internal
server error");
    }
}
```

Functional Details:

- `childService.getActiveChild()` : Links the therapy suggestion request to the active child

4.3 Get Pre-Signed URL

Endpoint: `GET /api/v1/aex/url/presigned`
Description: Generates a pre-signed URL for file upload.

Backend Implementation:

```
@GetMapping("/url/presigned")
public String fetchPreSignedUrl() throws Exception {
    return bucketStorageService.getPreSignedUrl("mp4");
}
```

Functional Details:

- `bucketStorageService.getPreSignedUrl()` : Generates a pre-signed URL for uploading a file to cloud storage

5. Drawing Feedback APIs

5.1 Compare Drawn image with Reference using Gemini

Endpoint: `POST /api/v1/drawing/compare_drawn_and_reference`
Description: Compares a user-uploaded drawing with a reference image.

Request Body:

```
{
  "image": "base64_encoded_image_data"
}
```

Backend Implementation:

```
@PostMapping("/compare_drawn_and_reference")
public ResponseEntity<?> handleImageUpload(@RequestBody ImageRequestModel request,
@RequestHeader("Authorization") String jwtToken) {
    try {
        String childId = childService.getActiveChild(jwtToken);
        if (childId == null || childId.isEmpty()) {
            return ResponseEntity.status(403).body("Invalid or missing token");
        }

        byte[] imageBytes = Base64.getDecoder().decode(request.getImage());
        Path destinationPath = Paths.get(uploadPath, UUID.randomUUID().toString() +
".png");
        Files.write(destinationPath, imageBytes);

        String feedbackOnImage =
geminiImageUploaderService.getImageFeedback(destinationPath.toString());
        return ResponseEntity.ok(feedbackOnImage);
    } catch (IOException e) {
        return ResponseEntity.internalServerError().body("Failed to save image: " +
e.getMessage());
    } catch (Exception e) {
        return ResponseEntity.internalServerError().body("An unexpected error
occurred: " + e.getMessage());
    }
}
```

Functional Details:

- `childService.getActiveChild()` : Retrieves the active child for associating the drawing
- `geminiImageUploaderService.getImageFeedback()` : Sends the image to an external service for feedback comparison

Explaination of Frontend methods.

Entire system is developed using VueJS.

UI Application Technical Documentation

Overview

This application is a Vue.js-based user interface for a medical assessment system, specifically focused on child development and autism screening. The application includes user authentication, child profile management, behavioral testing, and medical professional referrals.

Core Components

1. Authentication System

Location: `Authentication.vue`

Features:

- Dual-mode authentication (Login/Signup)
- JWT token-based authentication
- OTP verification system
- Cookie-based session management

Key Methods:

- `login()`: Handles user authentication and token storage
- `signup()`: Manages new user registration
- `toggleForm()`: Switches between login and signup forms

2. Child Profile Management

Location: `Home.vue`

Features:

- Multiple child profile support
- Active session management
- Profile switching functionality
- Automatic redirect to profile creation for new users

Key Methods:

- `fetchChildren()` : Retrieves all child profiles
- `switchProfile()` : Handles profile switching
- `activateChildSession()` : Manages active child session
- `handleSelect()` : Processes profile selection events

3. Behavioral Testing Interface

Location: `BehavioralVideoTesting.vue`

Features:

- Video upload capability
- Question-answer processing
- Automated analysis system
- Results routing

Key Methods:

- `fetchPresignedURL()` : Gets secure upload URL
- `uploadFile()` : Handles video upload
- `convertAnswersToText()` : Processes questionnaire responses
- `suggestTherapy()` : Generates therapy recommendations

4. Drawing Assessment Tool

Location: `DrawingGame.vue`

Features:

- Interactive canvas drawing
- Multiple drawing tools
- Image comparison functionality
- Audio feedback system

Key Methods:

- `startDrawing()` : Initializes drawing session
- `stopDrawing()` : Processes drawing completion
- `draw()` : Handles drawing operations
- `playPredefinedText()` : Manages audio feedback

5. Test Results Display

Location: `AexTestResults.vue`

Features:

- Confidence score display
- Q10 results presentation
- Video analysis results
- Child-specific result tracking

Methods:

- `mounted()`: Initializes result display
- Data fetching from API endpoints
- Result parsing and formatting

6. Medical Professional Referral System

Location: Final Vue component (unnamed in source)

Features:

- Location-based doctor sorting
- Rating-based doctor sorting
- IP-based proximity calculation

Key Methods:

- `fetchDoctors()`: Retrieves doctor listings
- `getUserIpAddress()`: Gets user location
- `onSortingChange()`: Handles sort preference changes

API Integration

Base URL

```
http://localhost:8080/api/v1/
```

Key Endpoints:

- Authentication: `/auth/login`, `/auth/register`
- Child Management: `/child/list`, `/child/add`
- Test Results: `/aex/lists`, `/aex/res/{id}`
- Video Processing: `/aex/url/presigned`, `/aex/invoke`
- Doctor Referrals: `/doctor/list/nearby`, `/doctor/list/ratings`

Security Features

Authentication:

- JWT token implementation
- Secure cookie storage
- OTP verification system
- Bearer token authorization

Data Protection:

- Presigned URLs for file uploads
- Session-based child profile access
- Token expiration handling

Technical Requirements

Frontend:

- Vue.js framework
- Axios for HTTP requests
- js-cookie for cookie management
- HTML5 Canvas API support
- Web Speech API support

API Dependencies:

- REST API endpoints

- JSON data format
- Bearer token authentication
- Multipart form data support

Error Handling

The application implements comprehensive error handling including:

- Network request error management
- File upload validation
- Authentication failure handling
- Session management errors
- API response validation

User Session Management

Session Features:

- Token-based authentication
- Active child profile tracking
- Multiple profile support
- Automatic session restoration
- Session expiration handling

Performance Considerations

Optimizations:

- Lazy loading of components
- Efficient state management
- Cached child profile data
- Optimized image processing
- Throttled API requests

Future Enhancements

Potential areas for improvement:

1. Implement WebSocket for real-time updates
2. Add offline support capabilities
3. Enhance error reporting system
4. Implement rate limiting
5. Add comprehensive analytics tracking

Reference


```
@InProceedings{Rani_2024_WACV,
    author    = {Rani, Asha and Verma, Yashaswi},
    title     = {Activity-Based Early Autism Diagnosis Using a Multi-Dataset
Supervised Contrastive Learning Approach},
    booktitle = {Proceedings of the IEEE/CVF Winter Conference on Applications of
Computer Vision (WACV)},
    month     = {January},
    year      = {2024},
    pages     = {7788-7797}
}
```