Pak. j. life soc. Sci. (2026), 24(1): 330-342 E-ISSN: 2221-7630;P-ISSN: 1727-4915

Pakistan Journal of Life and Social Sciences
www.pjlss.edu.pk

https://doi.org/10.57239/PJLSS-2026-24.1.0027

RESEARCH ARTICLE

Intelligent Face and Emotion Recognition System for Predictive Analytics of the Psycho–Emotional State of Students Participating in Military Conflicts in the University's Educational Environment

E.A. Revyakina[1]*, A.R. Gazizov[2]

[1,2]Don State Technical University, Rostov-On-Don, Russia

ARTICLE INFO

Received: Apr 24, 2025

Accepted: June 13, 2025

Keywords

Intelligent Video Analytics
Face Recognition
Emotion Analysis
Predictive Analytics
Psycho-Emotional State
Educational Environment
Participants in Military Conflicts
Computer Vision
Biometric Data

***Corresponding Author:**

revyelena@yandex.ru

ABSTRACT

The article addresses the problem of adaptation and support for students returning from military conflicts in the university educational environment. A solution based on an intelligent video analytics system is proposed, capable of real-time face detection, biometric template creation, facial expression analysis, and assessment of an integral indicator of psycho-emotional stress (aggression, anxiety, stress). The system is built on a modular architecture that combines neural network models Insight Face (face detection and embeddings), Deep Face and FER (emotion analysis), Media Pipe Face Mesh (geometric analysis of facial expressions) into a single video stream processing pipeline. Mechanisms for buffering and selecting the most informative frames are implemented, increasing the robustness of analysis in a dynamic educational environment. To ensure privacy and ethical application, a role-based access model (administrator, psychologist, teacher, student) and strict control over access to biometric data and media materials through secure APIs have been introduced. The developed web interface allows psychologists and curators to receive real-time notifications about cases of increased emotional stress, view event contexts, and conduct preventive work. The results demonstrate the practical applicability of the system as a predictive analytics tool for creating a safe and supportive educational environment for a special category of students.

INTRODUCTION

The modern higher education environment faces new challenges related to the integration of special categories of students, particularly students returning from war zones. This group is characterized by increased vulnerability and risk of developing post-traumatic stress disorder (PTSD), anxiety, depression, and outbursts of unmotivated aggression (Chollet, 2018; Kurakin et al., 2021). Traditional methods of monitoring psychological well-being (questionnaires, interviews) have significant limitations: they are reactive, require active student participation, and are unable to capture acute, situationally determined episodes of emotional stress that arise directly in classrooms, libraries, or campus public spaces.

The relevance of developing passive, unobtrusive, and predictive psycho emotional state (PES) monitoring tools in this context is obvious. Early detection of signs of escalating stress or aggression allows psychological services and study group supervisors to implement early intervention, provide targeted support, and prevent the escalation of crisis situations, including conflicts with fellow students or teachers, as well as self-injurious behavior (Ramenskiy, Lutsenko, 2020).

The aim of this work is to develop and validate the architecture of an intelligent system designed for predictive analytics of the psycho-emotional state of at-risk students based on the analysis of their facial and affective expressions in a video stream. The research focuses on methods and algorithms for real-time facial and emotion recognition, and the subject matter is the architecture and software implementation of such a system for the specific conditions of an educational environment.

The system is designed to solve the following problems:

1. Automatic detection and tracking of faces in the video stream from surveillance cameras installed in approved areas (e.g. halls, recreation areas, entrance areas).
2. Formation of anonymized biometric templates (embeddings) for anonymous recording of the frequency of occurrence and tracking the dynamics of the condition of a specific individual without direct identification of the person.
3. Multimodal analysis of facial expressions with assessment of basic emotions and calculation of an integral numerical indicator of the level of tension/aggression.
4. Creation of contextual notifications for responsible employees (psychologists) with strict control of access to visual data in accordance with ethical standards and legislation on personal data (152-FZ).

The development of any system that processes biometric data from vulnerable groups must be based on a solid ethical and legal foundation. The design of the proposed solution was conducted in strict accordance with a number of key principles. The principle of legality and appropriateness is based on the need to provide targeted psychological support within the educational institution, which is reflected in relevant legislation and the university's local regulations. Data minimization is achieved by eliminating the storage of redundant information: anonymized facial embedding is used instead of direct identifiers, and original video frames are stored for a limited period. The principle of "privacy by design" is embodied in the localization of data processing on the university server and the integration of security mechanisms into the system's architecture. Finally, the principles of transparency and selective access ensure that system users understand the logic of its operation, and access to sensitive information is strictly regulated by a role model (Kobtsev, Selivanov, 2019).

MATERIALS AND METHODS

The methodological basis of the system is the classical pipeline of facial analysis, supplemented by the tasks of emotion assessment and prediction. For the detection and formation of embeddings, the Insight Face framework was chosen, combining the modern Retina Face detector and the Arc Face architecture. Its advantages are high accuracy on complex data, an integrated processing pipeline, and efficient vector representations of faces. Embedding, acting as a digital "fingerprint", allows you to anonymously link events related to one person and build individual time series of status indicators.

The analysis of emotions is based on ensemble approach using two independent classifiers – Deep Face and FER (Ponomarev, Gritsenko, 2021). Using two models reduces the risk of error inherent in each of them individually. To improve explainability and add objective metrics, the Media Pipe Face Mesh geometric analysis module is used. It constructs a dense grid of key points on the face, allowing for the calculation of measurable features: eye opening, eyebrow and lip tension, and overall asymmetry. This makes it possible to move from the "black box" of the neural network to interpretable characteristics of facial expressions. The main identifier is the embedding – a numerical vector that describes unique facial features but does not allow for the reconstruction of the original photograph. This ensures anonymity when analyzing the dynamics of a person's state (Kharkhota, 2020).

The system's predictive component calculates an integrated psycho-emotional stress indicator on a scale from 0 to 10. This indicator aggregates the probabilities of "negative" emotions from classifiers and the values of key geometric features. To avoid false positives, threshold logic and analysis of indicator dynamics over a time window are used. It is important to emphasize that the final indicator is a signal for a specialist's attention, not a clinical diagnosis, and its thresholds are adjusted empirically during operation. The goal of the system being developed is to create an early warning tool for psychological support services.

Description of the General Structure of the System

The system's structure is built on a modular client-server architecture, ensuring scalability, security, and ease of support. The core of the system is a Python application server using the Fast API framework. It is responsible for receiving video data, executing the entire machine learning pipeline, handling business logic, interacting with the database, and providing APIs. PostgreSQL is used as the metadata storage system, storing information about users, devices, events, and access rights (Petrenko, Simonov, 2012). The original video frames and processed facial images are stored in

structured file storage on the server. For user interaction, a single-page web application was developed using React and Type Script, adapting its interface to the role of the logged-in employee. Real-time communication between the client and server is provided via the Web Socket protocol for sending notifications and receiving test video streams.

The algorithm of the system is the sequential processing of each video frame. It begins with receiving and authenticating data from a trusted source – an IP camera or a test client. After initial image processing, the Insight Face-based face detection module localizes all the faces in the frame, extracts an aligned crop for each of them, and generates embedding (Bobrova, Mastilin, 2021). Next comes the key stage of buffering. Instead of immediate analysis, the face's crop is placed in a buffer associated with its unique embedding. For each frame in the buffer, a quality score is calculated, taking into account sharpness and normalized brightness.

A background task periodically checks all active buffers. For those that have accumulated a sufficient number of frames, or the time limit has been reached, a single, high-quality sample is selected. This most informative frame is fed into the deep emotion analysis pipeline. Here, it is processed in parallel by two classifiers (Deep Face and FER) and the Media Pipe geometric analysis module. The resulting results - emotion probabilities and a set of numerical facial expression features - are aggregated into a final psycho-emotional stress score.

If the calculated indicator exceeds the threshold set by the administrator, the system generates an event. A record is added to the database with a timestamp, anonymous facial identifier, indicator value, metadata, and links to saved media files. The new event is immediately published to the Web Socket broadcast channel, and all active web clients subscribed to it with the appropriate permissions receive a real-time notification. If necessary, the notification can be duplicated to an external messenger, such as Telegram.

One of the important components of the system is the mechanism of secure media output (Panferov, 2022). Direct access to image files is not allowed. Instead, the web client requests media through a special server endpoint. With each such request, the server extracts the user's token, verifies his access rights to this particular event based on the role model and entries in the selective access table, and only after successful verification returns the file. This scheme ensures that even if links are leaked, access to confidential data will remain blocked for unauthorized persons.

Software Implementation of Key Algorithms and Modules

As a result of the analysis, four components of the local ML pipeline were selected in the work: Insight Face (face detection + embeddings), Deep Face and FER (two independent classifiers of emotions) and Media Pipe Face Mesh (geometry of facial expressions). The bundle covers the full cycle of frame analysis, preserves "contracts" (stable inputs/outputs of modules and data schemas), and the weights of any of the models can be replaced without rewriting the integration code.

Insight Face combines modern face detectors (e.g., Retina Face) and Arc Face-class embeddings. In everyday scenes (variable lighting, angles, partial occlusions), it demonstrates robustness and fast inference on CPU/GPU/ONNX/Tensor RT. Unlike dlib/Face Net/Open Face, which either have lower accuracy and robustness at complex angles or poorer support for accelerators and the ONNX format, Insight Face provides a ready-made production pipeline: alignment, embedding, comparison, and an API for the embedding database. Cloud-based alternatives (AWS Rekognition, Azure Face, Face++) offer strong metrics but are closed, charged per call, carry the risk of sharing biometrics with third parties, and are dependent on network/latency. For urban and retail scenarios where offline operation and local storage are important, the local Insight Face is preferable.

Face Mesh provides 468 anchor points Real-time, robust to moderate occlusions, runs on CPUs and mobile SoCs. Alternatives: 68-point dlib is simpler, but loses the nuances of facial expressions. Open Face/landmark models from Open VINO are comparable, but often require more complex assembly and produce lower mesh density (Revyakina, Gazizov, 2025). Media Pipe's geometry makes emotions "explainable": it can visually show which facial regions influenced the assessment. This is important for operator confidence and protection against accusations of a "black box."

Emotions are a more unstable task than identification, so two independent classifiers are used and aggregated with weights and post-processed based on frame quality. This improves robustness to

noise and dominant classes ("neutral," "happy"). Open-source alternatives - single models based on FER2013/RAF-DB/AFEW - typically generalize less well outside the domain or require significant additional training. Cloud SDKs (Affectiva/Smart Eye, Emotion as part of some Face APIs) provide convenient APIs, but they come with proprietary metrics/documentation, licensing costs, restrictions on on- premise use, and legal risks associated with sharing biometric data. The proposed ensemble design is transparent, extensible, and easily retrained using custom datasets.

Accuracy and stability: Insight Face + Face Mesh reliably process camera angles and lighting, and the emotion ensemble reduces false positives. Performance and portability: all components run locally, are compiled into ONNX/CPU/GPU, and support Tensor RT. Cloud-based processing offers better raw metrics, but comes with network/latency and billing considerations. Privacy and compliance: local processing without data export is a plus in addition to Federal Law No. 152 and internal policies. Cloud-based processing requires DPIAs/processing agreements and geo-residency.

Insight Face and emotions can be easily retrained using your own examples. Media Pipe provides stable features that can be used to train your own emotion/intent classifier.

Thus, the overall program algorithm relies on a combination of four neural network computer vision libraries: Insight Face, Deep Face, FER, and Media Pipe Face Mesh. Each library is connected The server-side frame processing pipeline is implemented using official Python packages (insight face, deep face, fer, media pipe) and receives the same normalized facial fragment as input (Redmon, Farhadi, 2018). All models are used in the form of ready-made weights published by the library developers; independent training was not performed as part of the final qualification work.

Insight Face was trained by the library's authors on large open datasets of facial images (MS1M-ArcFace, VGGFace2, CASIA-Web Face), ensuring high detection accuracy and stable embeddings. The FER module uses a pre-trained CNN model trained by the authors on the FER-2013 dataset, which includes 35,887 48x48 pixel facial images labeled with seven basic emotions. The Deep Face library integrates the VGG-Face, Face Net, and Arc Face architectures; the authors trained them using large SFC datasets (≈4.4 million images) and other open datasets of faces. These open-source models are used as the de facto standard for facial expression analysis.

Media Pipe Face Mesh is built on Google's proprietary architecture and trained on a multi-dataset annotation of facial landmarks - the specific sets are not disclosed, but the model is the industry standard for face tracking tasks. Insight Face generates an embedding - a compact numerical description of a person's appearance. This is compared with previously stored embeddings in the "faces" service table, allowing one to determine whether the observed person is the same or a new one. Deep Face and FER independently calculate a probability vector for basic emotions ("anger," "fear," "sadness," "joy," "surprise," "confusion," "neutral"), while Media Pipe produces the coordinates of 468 facial landmarks and derived facial features (eyebrow raise, eye opening, mouth opening, etc.) (Deng et al., 2019).

This data is combined into a single set of numerical characteristics. The risk assessment module then aggregates the results using thresholds and frame quality adjustments. The result is a severity score on a scale of 0 to 10, reflecting a combination of signs of tension, fear, anger, or abnormal facial expressions.

Server Architecture and Modules

The server component is implemented using Fast API with a clear separation of routes, service logic, and data access. The HTTP endpoint layer is organized by functional areas: authentication and token management, user accounting and device registration, frame reception and event creation, media delivery and visibility management, and administrative settings. An ORM approach is used for database interaction for the main entities - users, devices, tokens, and notifications - while the face embedding table is maintained separately, consistent with the purpose and specific nature of storing binary vectors. This asymmetry is deliberate: it separates "hot" business operations from specialized operations with feature representations (Ekman, Friesen, 1978).

Authentication is based on the JWT "access/refresh" pair. Upon login, the user receives a refresh token, which is stored in the database as a cryptographic hash, so a database compromise prevents the user from directly using refresh. Token rotation ensures session predictability and simplifies

maintenance (Deep Face, n.d.; Insight Face, n.d.). Devices are authenticated differently: upon registration, each is assigned an identifier and secret; when accessing the frame reception path, the server verifies the headers and allows only trusted sources. This separation of authentication paths eliminates the confusion between human and technological roles and allows for the establishment of distinct policies.

Routes are grouped into API versions. The basic version includes: login and token refresh, retrieving a list of users (for the administrator), device registration (also for the administrator), receiving a frame from a device, retrieving a list of events based on roles and permissions, visibility and address access management, secure media file delivery, and administrative settings, which, among other things, configure messenger integration parameters for notifications. Since image delivery is handled through a separate controller, all photo viewing requests undergo ownership and explicit permissions checks: the administrator has full access, the employee has access within the assigned permissions, and the citizen has no face unless a separate permission is granted.

The real-time channel is implemented as a Web Socket event route. A dedicated middleware layer operates at the connection input, extracting a token from the connection parameters, validating it, and binding the session to a role. The server publishes new notifications specifically; if an event is hidden or not supported by a given role, it is not included in the stream. This approach reduces the load on the client and server compared to constantly polling REST endpoints and ensures instant delivery.

Data is stored in a relational database: users, devices, token updates, notifications, and settings. The notification table contains fields for time, sources, coordinates, JSON structure of emotions, media links, and the "hidden" flag. A separate access table is maintained, specifying who is allowed to view what. Vector representations of faces are stored in a separate table with binary storage, which also stores links to a sample file for debugging or subsequent verification under legal circumstances. This structure allows for the isolation of highly sensitive data and minimizes the risk of unauthorized access.

The computer vision pipeline is encapsulated in a service module. It is responsible for the sequential steps of frame processing, selecting the best fragment based on quality, and generating structured conclusions. The threshold logic is based on a numerical severity indicator; when this indicator is exceeded, a database entry is created and a Web Socket publication is initiated. If the system settings allow for sending notifications to an external messenger, the handler uses the channel ID specified in the administrative section to transmit a brief event summary.

Client, Roles and Interfaces

The client-side component is built on Vite /React/Type Script and implements role-based navigation in a single application. Three main views correspond to roles: administrator, employee, and citizen, with an additional page configured for devices sending webcam footage. Authorization begins with a login form: the user enters credentials and receives a token, which is stored in the browser's storage and refreshed upon expiration. After login, the client determines the role and displays the corresponding sections.

The worker's dashboard focuses on a real-time event feed: cards contain the time, source, severity indicator, and, if permissions are available, a preview image. The secure media loading component accesses a secure server route via a link, passes a token, and waits for a response. If the user does not have permission to see the face, the component displays a limited version. Simple viewing and navigation scenarios are implemented between cards, allowing quick access to the most recent events. A map is also available, displaying events by coordinates, facilitating situational awareness.

The administrator has access to everything needed to manage the system: user and device registries, a list of events, including hidden ones, and a settings section. These settings allow for, among other things, external notification modes and channel identifiers. Actions are available to hide or explicitly allow viewing of individual events, including granting face access to specific users. Depending on the scenario, a citizen sees un faced views and is not allowed to perform operations that alter the system state.

The access control system is implemented in a dedicated server route for serving media files. When a client requests an image, the server extracts the event ID and file type (original frame or face crop) from the route. A multi-stage verification process follows. First, the user's JWT token is validated and their role is determined. The administrator gains access to all files automatically. For the psychologist, the logic is more complex: access to the original frame can be granted if the event was generated by a camera within their area of responsibility; access to the face crop requires either the same condition or an explicit entry in the selective access table, which the administrator creates in advance. The student role does not provide access to media through this mechanism. Only after passing all checks does the server generate a response with the requested file.

Data, Security and Rights

The system's data security policy is implemented technically and is applied throughout all image processing stages. Event images are never served directly: every request first enters a dedicated server route, which verifies step-by-step who is requesting the file and their permissions. The administrator can view all events, including hidden ones, and, if necessary, grant targeted access to specific materials. The duty officer works within their area of responsibility and only with those events to which they are explicitly granted access. A secure view without the portrait portion is generated for the citizen. The full frame, including the face, is shown only with special permission. The decision regarding who can see what is recorded in the service access table, so when serving an image, the server only needs to perform an unambiguous check of the entry in this table. The image files themselves are organized to reduce the risk of repeated unauthorized use: the original frames and facial fragments are stored in separate directory branches, and the full paths are never included in the client code or displayed in the user interface.

Human authentication and device trust are separated. Users are assigned a pair of electronic "passes": a short-term one for normal use and a longer one for extending the session. The second "pass" is stored in the database in encrypted form and cannot be used directly, even if the storage is compromised. Devices, on the other hand, are identified by their headers: upon registration, they are assigned an ID and a secret, which are verified on the server when a frame is received. As a result, neither the user nor the device can "jump" to the other's role or access the other's data solely by spoofing their parameters.

Digital facial fingerprints are stored as binary vectors, which make it impossible to reconstruct the facial image. In this form, they are useful only for comparing different events and for statistical calculations. Any operations with these vectors do not automatically generate images: access to media materials is still controlled through a rights table and individual permissions. This approach enables anonymized analysis by preserving links to original frames and facial fragments, but opening them only with legal grounds and explicit access rights. Media deletion and rotation policies are configured in the administrative section.

Description of the System Architecture

The system is built using a classic client-server architecture with a clear division of roles between the web client, application API, and computer vision module. The client application is responsible for authorization, interface management, and secure media retrieval. The server provides RESTful interfaces for managing users and devices, receiving frames from sources, emitting events and media data, as well as Web Socket channels for streaming and real-time notifications. The image analysis pipeline is integrated into the server and is used for both HTTP requests and incoming Web Socket streams. Data and processing artifacts are stored in a database and file structure, with access to the primary images managed through secure endpoints with permissions checks.

The client uses a compact Type Script application with a Vite build. The interface is divided into several modes depending on the user's role: administrator, worker, and citizen. After logging in, the token is saved in local Storage and applied to all requests. The administrator section displays an event summary with filters and a map: The Leaflet library is used to visualize coordinates, and event cards include links to protected media. For ease of administration, event operations are provided: hiding (hidden) with an indication of the reason and selective access sharing (granting permission to a specific user to view the face associated with a given event). These actions call the corresponding server endpoints and are immediately reflected in the interface. Worker mode is focused on

operational work with the feed and quick image viewing, without administrative actions. Civilian mode has limited rights: raw footage and most service information are not accessible, and access to the face is regulated by server-side rules. The client includes a dedicated page for testing an IP camera simulation: the component takes a video stream from a webcam, scales the frame to a reasonable size, maintains the aspect ratio, and regularly sends JPEG images to the server via the ingest REST interface. This mode allows you to test the entire processing chain and access policies without using real hardware.

Image transmission and display in the interface are implemented with security in mind: The Secure Media component never directly exposes files from a static folder, but requests content via a secure URL with a token, receives a binary stream, and displays it via a temporary object link in the browser. This eliminates the risk of compromised permanent links and prevents unauthorized downloads. The project also includes a utility component for user selection (for assigning event rights), which accesses an administrative list and returns a filtered set of candidates.

The server component is implemented using Fast API and provides a set of versioned handles under the /api/v1 prefix. The logic is grouped into areas: user authentication and management, device registration and inventory, frame reception, media output, administrative settings, and advanced event handling. Authentication is based on token pairs: a short-lived access token and a long-lived refresh token. Device passwords and secrets are stored as hashes using robust schemes. Users have roles such as admin, worker, and citizen, and devices have types (e.g., ip_camera and worker mobile), which are reflected in the rights verification logic. Devices are registered via an administrative handle, with a secret generated on the server side. This secret is displayed once upon creation and then stored only as a hash. For any request involving device operation (e.g., loading a frame), the ID and secret are transmitted as headers. The server checks for compliance and only then accepts the data. For user sessions, the Bearer scheme is used in the authorization header.

Visual data reception is supported in two ways. First, the ingest/frame REST route is available: the body contains the image in base64 format and, if available, geodata and auxiliary metadata (source size, sent frame size, etc.). Second, the /was/stream Web Socket channel is open, which accepts binary image fragments and associated information. This approach is convenient when it is easier for the source to maintain a persistent connection. Both methods feed the frame into a single internal pipeline: the image is added to a shared buffer using a specific face ID, and further steps are performed by a background worker. This approach eliminates code duplication and ensures consistent behavior regardless of the transport. In addition to the ingest channel, a separate /ws/alerts Web Socket is implemented for notification subscribers: as soon as an event with an aggression score above a specified threshold occurs, the server generates a compact JSON packet (time, event ID, score) and broadcasts it to all active connections. Authentication for both WS-routes is organized through a middleware layer that extracts and validates a token from request parameters or headers and, if necessary, restricts the list of allowed roles.

The core of image processing is located in a module called by the server and operates identically for both REST and WS. When a frame arrives, the server passes it to the face detector, receives a bounding box and a prepared facial fragment, as well as a numerical description (vector). This fragment and the face identifier are written to the observation buffer. The buffer is designed as follows: several recent crops for each face are stored, and for each, a suitability indicator is calculated - a combination of a sharpness assessment (based on the variance of the Laplace operator) and adequate brightness. Based on these values, the most informative frame is selected for further emotion analysis. This mechanism eliminates typical problems in a live scene - random blur, localized shadows, or overexposure on part of the face - without complex retrying or increasing the load on the detector. A background task checks at short intervals whether the buffers are ready: as soon as the window closes or a clearly best image has accumulated, the system launches emotion detection (categories and aggregated ratings) and generates the final result for recording. A separate artifact storage service is also provided: the original image (if transmitted) and the localized facial fragment are stored in a structured file hierarchy, and an event record with file links, coordinates (if any), ratings, and any additional metadata is added to the database. The aggression alert threshold is configured via an environment variable; when exceeded, a notification is triggered via WS, and if integration is enabled, an image with a caption is sent to a Telegram chat (the integration status is stored in the system settings table and is verified before publication).

Media data delivery is organized through a specialized controller that checks the user's role and, if necessary, access rights to a specific event. The principles are simple: raw images are not shown to civilians; an employee sees a face if they are the owner of the source device or the creator of the event; otherwise, only with explicit permission; the administrator has full access. To control selective access, an additional table is implemented, storing event-user pairs with a flag indicating whether the face can be shown. The administrator can enable this permission through the interface and also hide the event from the general feed (the "hidden" field). The interface optionally records the reason for this, which is included in the entry's metadata. On the client side, Secure Media relies on this controller: before opening a file, the component requests the content via a secure URL (token authentication) and displays the image in a pop-up window.

Domain data is represented in the database as a set of entities. The user model includes an identifier, email, name, role, and activity flags. The device model stores the name, type, secret hash, owner association, and auxiliary fields such as coordinates or zone. An event contains a timestamp, device association and, if applicable, the initiating user association, severity level, category label, face ID, emotion results in JSON format, links to raw and facial images, and arbitrary metadata. For selective access, there is a face viewing rights table (alert access) with unique event-user pairs. A separate system settings table records integration parameters (e.g., notifications enabled and chat ID). For refresh token registration and rotation, a storage model is provided for their hashes with expiration dates. When a pair is updated, the old hash is marked and a new one is generated. Database access is handled via ORM, and migrations include adding event hiding, ACL tables, and system settings - enough for stable operation without breaking existing data.

The set of external APIs has been simplified and is suitable for integration. Login and token refresh are available for authentication, while a user list and device operations (registration with secret issuance) are available for administration. For frame reception, a single ingest/frame method accepts basic JSON and, with the flush flag, can immediately flush buffers, which is useful in targeted sending scenarios. For media, a single handle returns the required file according to verified permissions. For the event display, an extended endpoint returns a list of calculated fields (including ready-made secure links), as well as share and hide/show operations. Both the admin interface and client pages for officers and civilians are built on top of this set.

Client Application

The client-side component is built on React and Type Script with a minimal set of dependencies and code organization by role. It is based on the standard Vite build stack. The development server configuration includes proxying of REST requests /api and Web Socket traffic /ws to the backend address, as well as enabling HTTPS in dev mode via the @vitejs/plugin-basic-ssl plugin. This is convenient for working with the camera in the browser: most devices require a secure context to access get User Media. The vite. config. ts file specifies a proxy scheme for http://localhost:8000 and separate rules for /ws/alerts and /ws/stream. Tailwind is used as a layer of utility classes, allowing for a neat and consistent page layout throughout the application without heavy UI libraries. The entry point to the interface is index.html with the root App component mounted from src/pages/App. tsx.

The authorization form is implemented as a sleek Login. tsx component: two input fields (email and password) pre-filled with test values for development (admin@example.com / test), a single Login button, and clear error indication (a red block message if the response is unsuccessful). When submitted, api. login is called, the access token is retrieved from the response, the user's role is extracted from the JWT payload, after which the token is saved in local Storage and passed up to App, where the appropriate mode (admin, employee, or citizen) is immediately opened. All fields and the button are styled with Tailwind utility classes, event handlers are protected from re-submission, and subsequent REST calls are made with the Authorization: Bearer header, ensuring a continuous session without unnecessary requests to the server.

The login screen is simple and functional: the user is presented with four buttons for explicitly selecting a mode - administrator, worker, citizen, and a test IP camera for quickly sending frames to the server. The server calls themselves are grouped in a single module, src/api.ts: it includes functions for logging in, working with the event list, managing users and devices, retrieving and

saving administrative settings, as well as separate methods for maps and test frame reception. A single point of access to the API reduces the likelihood of inconsistencies and simplifies maintenance.

The administrator role is contained in the src/pages/Admin. tsx component and provides a full set of functionalities without overwhelming the interface. The top panel includes switches between the Notifications, Users, Devices, Settings, and Map tabs. Each tab is an independent subcomponent, but they all share common principles: clear loading states, clean tables with a fixed header, short and clear action buttons, and a consistent approach to error handling.

In the event list, the administrator receives recent entries in reverse chronological order. Filtering is implemented across multiple axes: the minimum severity threshold is selected from a drop-down list, the search bar simultaneously checks for matches in device, user, shortcut, and address identifiers, the "Hidden" flag is enabled if necessary, and the "Since" field specifies a lower time limit using the native date time-local. This set covers everyday viewing scenarios without forcing users to switch between screens. Each entry in the table displays key fields and a short summary of emotions. Secure viewing buttons are associated with the media: The Secure Media component (src/components/Secure Media. tsx) does not provide direct links, but on click requests a file with an authorization header, converts the response to a Blob, creates a temporary object URL, and displays the image in its own modal window. When closed, the URL is correctly released via URL. Revoke Object URL, and page scrolling is blocked. The modal window is controlled in a predictable manner: closing it is done by clicking on the background or by pressing the Escape key. This solution is both convenient and secure: the actual storage paths are not disclosed, and viewing permission is verified by the server on every request.

Two actions are particularly important for operation and are explicitly implemented. The first is selective face access: the "Share" button opens a window with User Picker (src/components/User Picker. tsx), where you can quickly find a user by email or name substring and grant them access to a specific event. There's a separate "Show face" checkbox, which sets the access rule specifically for face crops. The second is event visibility control: the "Hide/Show" button toggles the flag and prompts you to specify a reason when hiding. The reason is saved next to the entry and displayed in a tooltip, which is convenient for auditing. Both operations are instantly reflected in the interface by a local patch of the entry array (patch Alert Local) and are then confirmed by a server response, reducing latency and improving user experience (Figure 1).

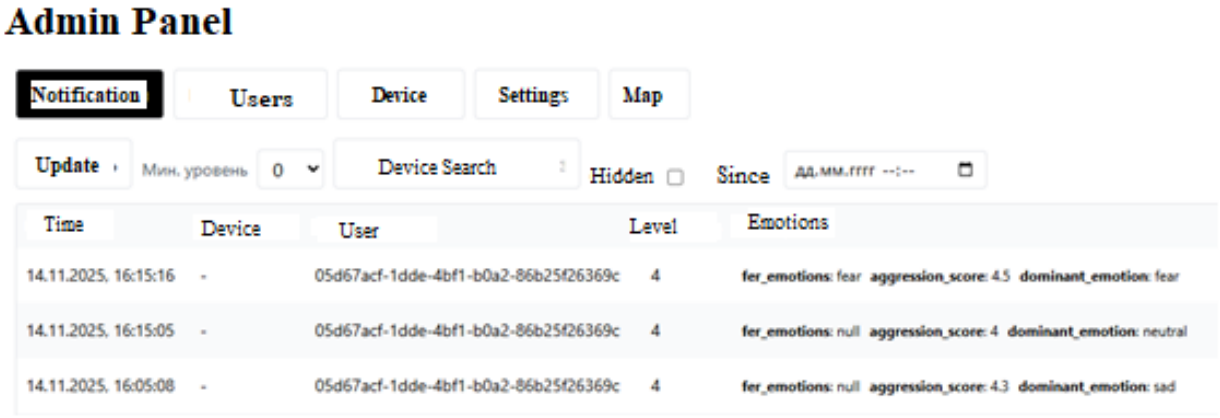

Figure 1: Administrator notifications page

The "Users" tab handles typical administrative tasks: displaying the current list with roles and IDs, creating new entries via a modal window, and deleting them if necessary. When creating a new entry, you specify an email address, password, and role (admin/worker/citizen). All statuses - loading, operation in progress, and errors - are displayed next to buttons, helping you stay organized during collaborative work. The "Devices" tab supports the full cycle of personnel source accounting. The list of devices is displayed in a table, with a button for registering a new device: a modal window prompts you to specify the device type (ip_camera or worker mobile), a user-readable name, and optionally select the owner via the familiar User Picker. After successful registration, a separate window displays the device_id and device_secret pair. The secret text is displayed only once, highlighted, and is accompanied by "Show/Hide" and "Copy" buttons. Below, the interface clearly reminds you that

these values should be transmitted by the device in the X-Device-Id and X-Device-Secret headers. This scenario exactly corresponds to the server side and removes most of the typical issues when connecting sources.

Administrative settings include managed integration with Telegram: you can enable or disable notification sending and set a chat ID. The "Save" button saves settings via a corresponding REST call, and an explanatory caption reminds you that flag checking and sending are performed on the backend, while the interface only sets the state. A "Map" tab has been created for situational assessment. The first screen displays a table of aggregated points with coordinates, average severity, number of events, and the last time. Next to it are aggregation parameters: coordinate rounding accuracy and lower time limit. Below is a true interactive map in Leaflet (react-leaflet): markers are circles whose size increases as the square root of the number of events, and their color ranges from green to red along the severity scale. When changing filters, the data is re-queried, and the map automatically adjusts the visible area to the set of points, eliminating the need to manually search for them. Adaptive padding keeps markers within the field of view even in dense layouts. The legend in the upper corner indicates the color correspondence to the severity level and reminds us of the dependence of size on the number of events.

The "worker" role is divided into two practical screens: "My Notifications" and "Stream." My Notifications" displays only those events to which the worker is assigned based on their roles and permissions. Filters are kept simple: there's a minimum severity level, a device/person search bar, an auto-update toggle, and a "Live" checkbox for connecting to the persistent notifications channel. When Live is enabled, the component opens a Web Socket to /ws/alerts with the user's token and processes incoming messages as soon as they appear. For robustness, a lightweight watchdog is included, ensuring the connection is closed when Live is disconnected. Media viewing is organized as securely as in the admin table: a separate use Secure Object Url hook replicates Secure Media's logic and loads the pop-up only when a modal window with details is open. Before this, there's no unnecessary network traffic, and the allocated memory is cleared when the window is closed. A modal window displays two images - a full-frame shot and a facial crop - and next to them, summary fields of the recording: time, source, IDs, level, basic emotions, and coordinates, if available. This approach combines a live feed with a detailed, on-demand window to avoid overloading the channel and wasting browser resources (Figure 2).

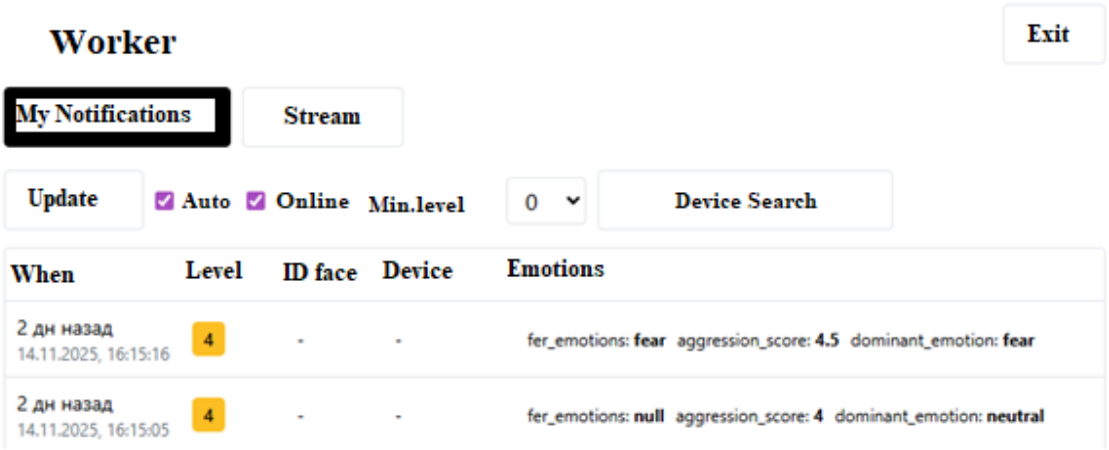

Figure 2: Employee notifications page

The "Stream" screen solves two practical problems: quickly connecting a broadcast from a worker's workstation and checking the entire path from the browser to the server. The page contains a <video> element with the correct auto Play, muted, and plays Inline attributes, ensuring correct operation on mobile devices, as well as "Start/Stop" buttons. When launched, the component requests access to the camera (get User Media), draws the current frame onto a temporary <canvas>, encodes it as JPEG, and sends it to the server via the /ws Web Socket channel with a one-second interval. Geolocation is simultaneously enabled: if permission is granted, the browser adds coordinates and an accuracy rating to frames, and the server can use this data when generating an event. The implementation is emphatically resource-efficient and behaves predictably: timers are

cleared when stopped, event handlers are removed, and the video stream is closed to prevent the camera from remaining active unnecessarily. This allows Stream to be used both for demonstration purposes and in workflows where it is important to quickly convey a situation into a general outline.

A separate IP Camera (Test) screen is designed to test REST frame reception (/api/v1/ingest/frame) and emulates an external source. The component (src/pages/Device Page.tsx) provides input fields for device_id and device_secret, optional lat/lon, and a set of useful buttons: turn on the camera, rotate the camera, and start sending. Camera switching is designed with real-world conditions in mind: if the device has two or more physical cameras, precise selection by device Id is used via enumerate Devices. Otherwise, the facing Mode switcher switches between the front and rear cameras. Before sending, the frame is neatly arranged in a square up to 720x720 while maintaining aspect ratio, encoded as JPEG with a reasonable quality factor, and a technical tag with the original and sent sizes is added to the JSON for debugging purposes. The sending period is one and a half seconds, and all server responses are displayed in a small log panel directly below the video stream, making it easy to monitor progress without the developer console. If the application is opened via HTTP outside of localhost, the page clearly warns that the browser will not allow access to the camera without HTTPS and offers an example frontend address using a secure protocol.

The citizen role is implemented as a lightweight event showcase (src/pages/Citizen's) without unnecessary actions and in compliance with access restrictions. It includes minimal filtering and the presentation of emotions as short key-value pairs for the first points, without attempting to reveal protected images.

This approach is suitable for scenarios where the event itself and brief context are important, but extensive media distribution is unacceptable. The entire page code follows the same principles as the other roles: clear loading indicators, error handling, clean tables, and the sparing use of state.

The component base is organized around several stable attributes. Modal (src/components/Modal. tsx) is a container with a background, title, and content area. It is used for user creation, device registration, and deletion confirmation. Secure Media, as noted above, handles all the tasks of secure media viewing: request generation, Blob loading, temporary URL creation, display in a modal, and closing via ESC. The src/ws/alerts. ts helper module simplifies subscription to the /ws/alerts channel: it accepts a token, opens a connection, and normalizes incoming messages, returning a pre-parsed object. This separation keeps pages short and readable: each performs its own role, and reusable logic is separated into separate modules.

Overall, the client application does exactly what's needed in an applied recognition system: it allows the administrator to manage users and sources and control the event flow, allows officers to quickly view their notifications and go live when needed, allows citizens to easily receive information within their rights, and allows integrations to be configured centrally. The interface is uncluttered, yet it incorporates important details that play a significant role in operation: secure media display without direct paths, predictable filters, and secure camera and geolocation handling in the browser. All of this complements the server component and makes the system seamless: from device connection and user login to event viewing and prompt notification, there's a single logical flow, with every element in its place.

RESULTS AND DISCUSSION

The developed service performs automatic facial recognition and emotion analysis in video streams. Its main tasks are to identify faces in frames, create embeddings for identification, determine basic emotional states, and calculate an integrated aggression index. Upon detecting increased levels of aggression or disturbing behavior, the system generates an alert and transmits it to users via a web interface and messenger notifications. Thus, the program functions as an intelligent monitoring module, enhancing the information yield of traditional video surveillance.

Furthermore, the service has potential for application in a number of areas. First and foremost, it can be used in video surveillance systems for shopping malls, train stations, airports, and other public gathering places, where it's important not only to record incidents but also to predict their development based on emotional state analysis (Media Pipe Face Mesh. n.d.; Py-Feat, n.d.). Integration with existing security software (e.g., corporate access control systems, situation centers, and security consoles) will expand their functionality through automated analytics.

The service can also be applied in related areas, from smart cities and municipal surveillance cameras to transport platforms and educational institutions, where it's crucial to promptly detect signs of aggression or disturbing behavior. Thanks to its modular architecture and use of standard APIs, it can be integrated into existing video surveillance software packages and into custom corporate solutions.

Thus, the created software is not limited to a research prototype, but has the potential for industrial application in security systems, analytical products, and intelligent monitoring services.

CONCLUSION

The completed work demonstrated the feasibility and feasibility of creating a comprehensive intelligent system for predictive analytics of psycho-emotional states in a university campus setting. The developed architecture ensures functional completeness, covering the entire cycle from video data reception to the generation of contextual notifications, and offers significant flexibility thanks to its modular design. The implementation of an adaptive buffering mechanism represents an important engineering solution, directly impacting operational stability in real-world, non-ideal conditions.

The system was designed from the outset with strict ethical and legal requirements in mind. Principles of processing localization, data minimization and anonymization, and detailed access control are embedded in its foundation, minimizing potential risks and forming the basis for trusted implementation. The solution's practical focus is confirmed by a user-friendly web interface for specialists, implementing key workflows: event feed monitoring, context analysis, and access management.

At the same time, it is important to recognize the fundamental limitations of the approach itself (Dzhurov et al., 2023; Kodatsky et al., 2024; PostgreSQL, n.d.). Facial expression analysis, even enhanced by geometric methods, remains probabilistic and ambiguous. Similar facial patterns can reflect different internal states, and individual characteristics of emotional expression can introduce systematic errors. Therefore, the system should be considered solely as a screening and decision-support tool, not as an automated diagnostic system. The final interpretation of the signal and the decision regarding intervention should always rest with a qualified psychologist.

Further development of the system is envisioned in several directions. A promising approach is to move toward multimodal analysis, including assessment of body posture, gestures, and paralinguistic vocal characteristics. The development of long-term monitoring modules capable of identifying not only acute episodes but also chronic trends in a student's condition holds great potential.

Thus, the proposed system represents a significant step toward the digitalization of psychological support in higher education. It demonstrates how modern artificial intelligence technologies, when ethically and thoughtfully integrated into organizational processes, can serve not as a control tool, but as a means of predictive support and care. Such a tool can enhance the capabilities of psychological services, enabling a transition from a crisis-response model to a preventative one, thereby contributing to the creation of a safer and more inclusive environment for all students.

REFERENCES

Bobrova MV, Mastilin AE. Machine learning in cybersecurity. In: Yemelyanov NV (Ed.), Nauchnyye Mezhdistsiplinarnyye Issledovaniya [Scientific Interdisciplinary Research]: Proceedings of the XIII International Scientific and Practical Conference. Moscow: "KDU", "Dobrosvet"; 2021. p. 24-9.

Chollet F. Deep Learning with Python. Moscow: Dialectika; 2018. 384 p.

DeepFace: Lightweight face recognition and facial attribute analysis framework. n.d. https://github.com/serengil/deepface (accessed on April 15, 2025).

Deng J, Guo J, Xue N, Zafeiriou S. ArcFace: Additive angular margin loss for deep face recognition. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New York, NY: IEEE; 2019. p. 4685-94. https://doi.org/10.1109/cvpr.2019.00482

Dzhurov AA, Cherkesova LV, Revyakina EA. Software tool for detecting fake video content using the Deepfake technology of the GAN algorithm. H&ES Res 2023,15:60-7.

Ekman P, Friesen WV. Facial Action Coding System: Investigator's Guide. Palo Alto: Consulting Psychologists Press; 1978. 52 p.

InsightFace: 2D and 3D Face Analysis Project. n.d. https://github.com/deepinsight/insightface (accessed on April 15, 2025).

Kharkhota SN. Information Security: Information Protection in Computer Systems. Moscow: Academy; 2020. 256 p.

Kobtsev VV, Selivanov AA. Biometric Technologies: Facial Recognition. St. Petersburg: Piter; 2019. 352 p.

Kodatsky N, Revyakina E, Gazizov A, Gunko V. Retracted: Using machine learning to forecast hard drive failures. E3S Web Conf 2024,549:08024. https://doi.org/10.1051/e3sconf/202454908024

Kurakin DV, Belyaev AN, Mitrokhin AS. Face recognition using convolutional neural networks. Inform Technol 2021,5:48-54.

MediaPipe Face Mesh. n.d. https://github.com/google-ai-edge/mediapipe/blob/master/docs/solutions/face_mesh.md?ysclid=mm6lhx77jf349729282 (accessed on April 15, 2025).

Panferov AM. Artificial Intelligence and Security: Algorithms, Neural Networks, Practice. Moscow: Forum; 2022. 304 p.

Petrenko SA, Simonov SV. Information Risk Management: Cost-Effective Security. Moscow: Akad. AyTi: DMK Press; 2012. 384 p.

Ponomarev DV, Gritsenko SM. Application of neural networks for the analysis of emotional state from a video image of a face. Probl Manag 2021,1:82-91.

PostgreSQL. Official website of the PostgreSQL DBMS. n.d. https://www.postgresql.org/ (accessed on April 15, 2020).

Py-Feat: Python Facial Expression Analysis Toolbox. n.d. https://github.com/cosanlab/py-feat (accessed on April 15, 2025).

Ramenskiy SYu, Lutsenko IA. Application of machine learning methods to problems of emotion analysis based on facial images. Bull Moscow Univ Ser 15 Comput Math Cybernet 2020,4:32-42.

Redmon J, Farhadi A. YOLOv3: An Incremental Improvement. arXiv:1804.02767v1 [cs.CV] 8 Apr 2018.

Revyakina E, Gazizov A. Development of a data link protection model based on algorithmic and hardware support. Pak J Life Soc Sci 2025,23:337-45. https://doi.org/10.57239/PJLSS-2025-23.2.00171