Technical Due Diligence

DFL DIGITAL SPORTS

Architecture and platform review:	
1.1. What technologies and frameworks are used?	Our platform utilizes a modern technology stack carefully selected for performance, scalability, and developer productivity: Backend Technologies: * Elixir/Phoenix: Primary language and framework for our core backend services, chosen for its exceptional concurrency model and reliability for real-time processing * Python: Used for AI/ML microservices and specialized data processing pipelines * PostgreSQL: Primary relational database with advanced partitioning strategies * ElasticSearch: Search engine for efficient text analysis and complex querying * Redis: In-memory data store for caching and real-time features Frontend Technologies: * React: Primary framework for modern UI components and interactive dashboards * TypeScript: Used for type-safe JavaScript development * Ember.js: Legacy framework for some existing components (actively being migrated to React) * Modern CSS with SCSS preprocessor * Recharts and D3.js for data visualization Infrastructure and DevOps: * Amazon Web Services (AWS) as our cloud platform: - ECS for container orchestration - RDS for managed PostgreSQL databases - S3 for object storage - CloudFront for CDN services - CloudWatch for monitoring - CloudAMQP for queue management in feedback * Docker for containerization * GitHub Actions for CI/CD automation * Terraform for infrastructure as code * Sentry for error tracking and performance monitoring AI and Machine Learning: * Azure OpenAI for advanced natural language processing * Custom Python ML pipelines for domain-specific analysis * TensorFlow for some specialized machine learning models * scikit-learn for traditional machine learning algorithms

	Integration and APIs: * RESTful APIs following OpenAPI specifications * GraphQL for some data-intensive client applications * WebSockets for real-time features * OAuth 2.0 and JWT for authentication Security Technologies: * AWS Security services (GuardDuty, Config, CloudTrail) * Automated vulnerability scanning * Dependabot for dependency security management * HTTPS/TLS for all communications This technology stack provides us with the performance and scalability needed for our data-intensive platform while enabling rapid development and deployment of new features. Each technology has been selected to address specific requirements, with particular emphasis on real-time processing capabilities, efficient handling of large datasets, and responsive user interfaces.
1.2. How is the platform built?	The zenloop platform is built using a modern, microservices-inspired architecture with the following key components: * Real-time data processing pipeline using Elixir for handling large-scale, multi-staged survey data * Modular, API-first architecture allowing integration with CRM, ESP, and personalization platforms * Machine learning pipeline for language detection, topic assignment, and sentiment analysis Technology Stack: * Backend: Implemented as an Elixir/Phoenix umbrella application with separate applications for different concerns, with Python used for specialized AI components * Frontend: React for modern UI components with some legacy components in Ember, hosted on Vercel * Database Layer: PostgreSQL serves as the primary transactional database with table partitioning for optimization * Search Engine: ElasticSearch for efficient text analysis and search capabilities across multilingual survey responses * Infrastructure: Deployed on Amazon Web Services (AWS) using ECS, RDS, ElasticSearch Service, and S3

	Microservices Architecture: The platform consists of specialized services handling distinct functions: * Survey management service * Rendering service * Analysis service * Feedback analysis service * Report creation service * User Access Management (UAM) Data Flow: * Survey responses are first stored in PostgreSQL (source of truth) * Then indexed into ElasticSearch for searching and analysis * Processing pipeline analyzes text, assigns topics, and extracts sentiment * Results are made available via RESTful APIs and dashboards Deployment and Infrastructure: * Services are containerized using Docker for consistent deployment * CI/CD pipelines implemented through GitHub Actions * Three environments: Development, Staging, and Production * AWS infrastructure includes ELB, ECS Cluster, RDS PostgreSQL,and S3 storage * Monitoring through CloudWatch, Prometheus, and Grafana ElasticSearch
1.3. Are there areas that are overly complex or prone to bugs?	While our architecture is generally well-structured, we do have a few areas of increased complexity that require careful management: 1. Synchronization between PostgreSQL and ElasticSearch: Maintaining consistency between our primary database and search index requires sophisticated handling of race conditions and edge cases, especially during reindexing operations. 2. Multi-language support: Our text analysis pipeline supporting numerous languages adds complexity to both the codebase and testing requirements. 3. Legacy Ember components: The coexistence of React and Ember in the frontend creates some integration challenges and increases the learning curve for new developers. However, we are already actively migrating all functionality to React, which will significantly reduce this complexity in the near future.

	4. Partitioning strategy: While our PostgreSQL partitioning approach improves performance, it also adds complexity to query optimization and requires careful management during scaling. 5. Integration complexity: The numerous third-party integrations (CRMs, ESPs, etc.) introduce potential points of failure that are outside our direct control. We mitigate these complexities through: - Comprehensive automated testing, particularly for data synchronization - Detailed developer documentation - Incremental approach to legacy code modernization - Regular monitoring of critical integration points
1.4. What language are you using?	Backend: * Primarily Elixir for our core backend services and real-time processing pipeline * Python for all our new AI-focused microservices, including language detection, topic assignment, and sentiment analysis * All new "smart services" leveraging artificial intelligence capabilities are consistently built using Python Frontend: * JavaScript/TypeScript with React for modern UI components and dashboards * Some legacy components in Ember (actively being migrated to React) * CSS/SCSS for styling Integration: * JavaScript for web widget integration * RESTful API with JSON for broader integrations with third-party systems * Elixir-based API services for handling client requests Query Languages: * SQL for PostgreSQL database operations via Ecto * Elasticsearch Query DSL for search functionality The main application architecture is built as an Elixir/Phoenix umbrella application, with separate applications for different concerns. This gives us the benefits of Elixir's concurrency model for handling large volumes of survey data while maintaining high performance. For AI and machine learning capabilities, we've strategically chosen Python to leverage its rich ecosystem of data science and machine learning libraries.
1.5. Do you support big frameworks like angular, react, vue?	Our platform's frontend uses React and some legacy Ember components internally. However, from an integration perspective:

	* Zenloop widgets and surveys are designed to be framework-agnostic. They can be embedded in any web application regardless of the frontend framework used (Angular, React, Vue, or others). * Integration is primarily achieved through: 1. JavaScript snippets that can be added to any webpage 2. RESTful API endpoints that can be called from any framework 3. Webhooks for event-driven interactions * For React applications specifically, we provide optimized integration patterns since our own frontend is primarily React-based. * Our widgets are designed with minimal dependencies to avoid conflicts with the host application's frameworks or libraries. This approach ensures maximum flexibility for our customers, allowing them to integrate Zenloop functionality regardless of their chosen frontend technology stack.
1.6. How is the technique behind a widget on iOS?	For iOS integration, Zenloop offers the following approaches: 1. WebView Implementation: * Our primary iOS integration method uses WKWebView to display responsive web-based surveys * The mobile-optimized survey is loaded via URL or HTML content * This approach maintains visual consistency with web surveys while working within native apps 2. API Integration: * For fully native experiences, our RESTful API allows developers to build custom UIs * iOS apps can use Swift/Objective-C to communicate with our backend services * This provides maximum customization but requires more development effort 3. Deep Linking Support: * Surveys can be triggered via custom URL schemes or universal links * This allows seamless transitions between app screens and survey interfaces * Supports continuation of user journey after survey completion 4. iOS SDK (Beta):

	* We're currently developing a native iOS SDK that will simplify integration * The SDK will handle authentication, caching, and offline functionality * It will provide native UI components that maintain Zenloop's functionality while following iOS design guidelines All iOS integration methods support the core features of our platform, including NPS collection, custom questions, and feedback analysis, while adhering to iOS platform guidelines and performance standards.
1.7. How is the technique behind a widget on web?	Our web widgets are designed for seamless integration with any website or web application through several technical approaches: 1. JavaScript Embedding: * Our primary method uses lightweight JavaScript snippets that customers add to their websites * The snippet asynchronously loads our widget library to minimize impact on page load performance * Widgets are rendered within iframes to prevent CSS/JavaScript conflicts with the host website * This approach supports responsive design and adapts to different screen sizes 2. RESTful API Integration: * For custom implementations, our comprehensive RESTful API allows developers to build fully customized survey experiences * API endpoints handle survey definition, response collection, and data retrieval * Authentication is managed via JWT or API keys 3. Direct Link Integration: * Surveys can be shared via direct links for email campaigns or social media * These links open the survey in a dedicated, branded page 4. Integration Partner Connectors: * Pre-built connectors for common platforms (e.g., Shopify, WordPress) * These connectors handle the technical implementation details automatically All web widgets are built with modern web standards, using a React-based frontend for interactive elements. They are designed to be lightweight and non-intrusive, with careful attention to accessibility standards (WCAG compliance) and cross-browser compatibility across all modern browsers.

Technical Due Diligence

DFL DIGITAL SPORTS

1.8. Which OS versions and browser do you support?	Our platform is designed to be accessible across all modern browsers and operating systems, with specific support as follows: Supported Desktop Browsers: * Google Chrome: Current and previous major version * Mozilla Firefox: Current and previous major version * Apple Safari: Current and previous major version * Microsoft Edge: Current and previous major version * Opera: Current version Supported Mobile Browsers: * iOS Safari: Last two major iOS versions (currently iOS 16 and 17) * Android Chrome: Last two major versions * Samsung Internet: Last two major versions Supported Operating Systems: * Windows: Windows 10 and 11 * macOS: Last three major versions (currently Ventura, Sonoma, and Monterey) * Linux: Major distributions with modern browser support * iOS: Last two major versions (currently iOS 16 and 17) * Android: Last three major versions (11, 12, and 13) Browser Feature Requirements: * JavaScript enabled * Cookies enabled for authentication * LocalStorage/SessionStorage for state management * Modern CSS support * TLS 1.2 or higher Accessibility Compliance (for surveys): * Designed to meet WCAG 2.1 AA standards * Screen reader compatibility with major assistive technologies * Keyboard navigation support * Sufficient color contrast ratios Our Terms of Service explicitly require users to maintain updated browsers for optimal security and functionality. While the platform may function on older browser versions, we officially support and test on the versions listed above. Our responsive design approach ensures consistent functionality across device types and screen sizes.

	For customers with legacy system requirements, we can provide guidance on minimum viable configurations, though we strongly recommend using modern, updated browsers for the best experience and security.
1.9. What are the limitations of the platform?	The zenloop platform has been designed to be robust and flexible, but there are some current limitations to be aware of: 1. Framework Integration: * While our widgets are framework-agnostic, some advanced customization options are more streamlined for React applications compared to other frameworks 2. Scale and Performance: * Our Elixir backend is highly optimized for concurrent processing, but extremely high-volume implementations (>15M responses/month) may require dedicated infrastructure configuration * For organizations with complex multi-region requirements, additional configuration may be needed to optimize performance 3. AI Feature Limitations: * Our new Python-based AI services provide powerful analysis capabilities but have some constraints: - Language detection and sentiment analysis currently support 15 major languages; less common languages have reduced accuracy - Topic clustering effectiveness can vary based on response volume (requires minimum thresholds for optimal performance) - AI-driven insights improve with data volume; smaller datasets may produce more general insights 4. Custom Integrations: * While we support a wide range of pre-built integrations, highly customized enterprise systems might require custom development work * Multi-step API workflows for complex use cases may require implementation support from our team 5. Data Processing: * The PostgreSQL partitioning strategy provides excellent performance but requires careful management during scaling * ElasticSearch synchronization with PostgreSQL requires sophisticated handling of edge cases We continuously address these limitations through our product roadmap, with current initiatives focused on: * Completing the migration to React for improved frontend consistency

	* Enhancing our AI capabilities to support additional languages and more sophisticated analysis * Expanding our integration ecosystem with new pre-built connectors * Optimizing our database architecture for even higher scalability
1.10. What is your roadmap?	Our product roadmap focuses on leveraging cutting-edge AI technologies and journey-specific capabilities to transform how businesses understand and act on customer feedback: 1. Advanced AI Customizations * Enhanced natural language processing capabilities for deeper insight extraction * Custom AI models that can be trained on industry-specific terminology and sentiment patterns * Personalized feedback analysis frameworks based on individual business priorities and KPIs 2. End-to-End Customer Experience Management * AI agents that autonomously monitor, analyze, and recommend improvements across the entire customer journey * Predictive analytics to forecast customer satisfaction trends and proactively identify potential issues * Automated workflow triggers based on feedback patterns to close the loop on customer concerns 3. Journey-Specific Survey Enhancements * Specialized survey templates optimized for specific customer journey stages (acquisition, onboarding, retention, support) * Dynamic question sequencing that adapts based on the customer's position in their journey * Intelligent touchpoint mapping to ensure feedback is collected at the most impactful moments * Cross-journey analytics to identify how experiences in one phase affect satisfaction in others 4. E-commerce Integration Expansion * Specialized product review analytics with direct integration to Google Shopping Ads and Bing Shopping Ads * Competitive benchmark analysis across major retail platforms * Review sentiment correlation with product performance metrics 5. Platform Enhancement * Completion of our migration to React for a more consistent and performant frontend experience

	* Scaling our Elixir-based processing pipeline for even higher throughput * Enhanced visualization capabilities for more intuitive reporting dashboards 6. Enterprise Capabilities * More sophisticated multi-tenant management for global organizations * Advanced role-based access controls for enterprise-scale deployments * Custom data retention and compliance tools for regulated industries Our development is guided by continuous customer feedback and market trends, with quarterly releases of major features and more frequent updates for enhancements and optimizations. We maintain our commitment to creating a platform that not only collects feedback but transforms it into actionable business intelligence tailored to every stage of the customer journey.
1.11. What are the SLAs (e.g. support model, availability, reaction time, service level credits, backup routines, RTO, RPO) of the platform?	Verfügbarkeit der SaaS-Dienste (Service Level Agreement – SLA) Der Anbieter stellt dem Kunden die SaaS-Dienste während der Vertragslaufzeit mit einer Verfügbarkeit von 99,5% (pro Kalenderjahr) bereit. Maßgeblich ist die Verfügbarkeit der SaaS-Dienste am Übergabepunkt des Systems zum Internet. Die Verfügbarkeit berechnet sich nach folgender Formel: Verfügbarkeit = (Gesamtzeit - Gesamtausfallzeit) / (Gesamtzeit * 100 %) Bei der Berechnung der Gesamtausfallzeit bleiben folgende Zeiten außer Betracht: Zeiten der Nichtverfügbarkeit wegen geplanter Wartungsarbeiten an der Plattform, die regelmäßig zwischen 18.00 und 22.00 Uhr abends (MEZ) durchgeführt werden, wobei die maximale Ausfallzeit pro Woche 1,5 Stunden beträgt. Zeiten der Nichtverfügbarkeit wegen Instandhaltungsarbeiten, die wöchentlich am Dienstag und Donnerstag zwischen 10:00 Uhr und 10:20 Uhr morgens (MEZ) durchgeführt werden. Zeiten der Nichtverfügbarkeit wegen geplanter Aktivitäten zur Verbesserung, Erweiterung oder Erneuerung der SaaS-Services, die am Wochenende ausgeführt werden Zeiten wegen zwingend erforderlicher außerplanmäßiger Wartungsarbeiten, die zur Beseitigung von Störungen erforderlich sind; zenloop wird den Kunden hiervon nach Möglichkeit durch einen Hinweis auf der Website in Kenntnis setzen; Zeiten der Nichtverfügbarkeit, die auf Störungen des Internets oder auf sonstigen von zenloop nicht zu vertretenden Umständen, insbesondere auf Höheren Gewalt beruhen; Zeiten der Nichtverfügbarkeit, die darauf beruhen, dass die vom Kunden zu schaffenden erforderlichen technischen Voraussetzungen für den

	Zugang zu den SaaS-Diensten nicht oder vorübergehend nicht gegeben sind, beispielsweise bei Störungen der Hardware des Kunden.
Software Development Process & Code review:	
1.12. Review of the platform's code. How clean is the code? Are there areas that are overly complex or poorly documented? How is the code maintained and updated?	Our codebase is architected with a strong emphasis on maintainability and clarity, though like many evolving platforms, it contains both newer, well-structured components and areas slated for modernization: Code Quality: * Our Elixir backend follows established functional programming patterns with clear separation of concerns * The codebase uses an umbrella application structure that provides well-defined boundaries between components * Our newer Python-based AI microservices are implemented following modern best practices with comprehensive test coverage * Frontend code has varying levels of consistency, with newer React components following strict style guidelines and older Ember components scheduled for migration Documentation: * All new code requires comprehensive documentation as part of our pull request process * We maintain detailed architecture documentation that outlines system components and their interactions * API endpoints are documented with OpenAPI/Swagger specifications * Our Elixir backend includes extensive module and function documentation * Python services include docstrings and usage examples Areas for Improvement: * Some older components in the Ember frontend have less comprehensive documentation than newer React components * Certain complex data processing flows, particularly around ElasticSearch synchronization, could benefit from more detailed documentation * The transition between our Elixir backend and Python AI services introduces some complexity that we're continuously refining Maintenance and Update Process: * We follow a structured Git workflow with feature branches and pull requests * All code changes require peer review before merging * Automated CI/CD pipelines run comprehensive test suites for every change

	* We use static code analysis tools to maintain code quality standards * Regular refactoring initiatives address technical debt in a systematic way * Weekly deployments for feature updates and bug fixes, with critical security patches deployed as needed * A formal deprecation process ensures smooth transitions when replacing legacy components We are continuously improving our codebase through several initiatives: 1. Ongoing migration from Ember to React 2. Enhancement of our automated testing coverage 3. Standardization of documentation practices across all components 4. Modularization of complex systems to improve maintainability
1.13. What is your software development process like (requirements gathering, development, testing & deployment)?	We follow an Agile development methodology centered on customer value creation, with a continuous feedback loop that ensures our platform evolves in direct response to user needs: Requirements Gathering: * Customer-centric approach starting with user interviews and feedback analysis * Regular customer advisory board sessions to identify high-impact feature opportunities * Data-driven prioritization using metrics on feature usage and customer pain points * Cross-functional workshops involving product, engineering, design, and customer success teams * Value-stream mapping to understand how features translate to tangible customer outcomes * Creation of detailed user stories with clear acceptance criteria and measurable success metrics Development Process: * Two-week sprint cycles with daily stand-ups for team synchronization * Feature-based development teams with end-to-end ownership * Trunk-based development with feature flags for progressive rollouts * Pair programming for complex features to ensure knowledge sharing * Code reviews required for all changes with automated quality checks * Internal developer documentation maintained alongside code changes * Technical debt addressed through dedicated refactoring sprints (20% development time) Testing Strategy: * Comprehensive automated testing pyramid: - Unit tests for individual functions and components

	- Integration tests for service interactions - End-to-end tests for critical user journeys * Continuous Integration with automated test runs on every commit * Exploratory testing sessions for each new feature * Internal dogfooding where our own teams use new features before release * Beta programs for selected customers to provide early feedback * Performance testing for high-impact changes to ensure scalability Deployment Process: * Continuous Deployment pipeline using GitHub Actions * Staged rollout approach: - Development environment for active work - Staging environment for integration testing - Production environment with canary deployments * Automated database migrations with rollback capabilities * Feature flags to control feature visibility and progressive rollouts * Post-deployment monitoring with automated alerting * Deployment frequency of once to twice a week for feature updates Value Validation: * Post-release metrics tracking to measure feature adoption and impact * A/B testing for UI/UX changes to quantify improvements * Regular retrospectives to capture learnings and refine processes * Feedback loops with customers who requested features to validate solutions * Quarterly reviews of feature performance against success metrics * Adjustments based on real-world usage patterns and customer feedback Our development process emphasizes flexibility, transparency, and continuous improvement. By maintaining close alignment with customer needs throughout the development lifecycle, we ensure that every feature delivers measurable value while maintaining the stability and performance of our platform. .

Technical Due Diligence

DFL DIGITAL SPORTS

1.14. What is your team structure like including roles and responsibilities, level of experience and expertise and collaboration processes?	Our team structure is designed for agility and technical excellence, with a lean, highly experienced group organized to maximize both innovation and operational efficiency: Team Composition: * Full Stack Engineers (core development team) with significant expertise: - Minimum 6 years of professional development experience - Average industry experience of 8-10 years per engineer - Diverse technical backgrounds spanning enterprise SaaS, data processing, and AI systems - Deep expertise in our core technologies (Elixir, Python, React, PostgreSQL, ElasticSearch) * Product Management: - Product Manager responsible for roadmap prioritization, customer research, and feature specification - Works closely with engineering to translate customer needs into technical requirements - Owns the product backlog and manages stakeholder expectations * Implementation Architecture: - Implementation Architect who bridges technical and business domains - Responsible for solution design, technical integration planning, and system scalability - Works directly with enterprise customers on complex implementations - Provides technical guidance to ensure solutions meet customer requirements * Customer Success: - Customer Success Managers who gather feedback and ensure customer satisfaction - Collaborate with the product team to identify enhancement opportunities - Deliver training and ensure customers maximize platform value - Feed insights back to the development team to guide future improvements Collaboration Process: * We follow a streamlined Scrum methodology optimized for execution speed: - Two-week sprints with clearly defined goals and deliverables - Daily stand-ups focused on progress, blockers, and coordination - Sprint planning sessions that emphasize customer value and technical feasibility - Retrospectives to continuously refine our process

	- Demo sessions where completed work is presented to the wider team * Cross-Team Collaboration: - Weekly team-wide demos where all teams present their work to the entire organization - These demos enhance cross-team awareness, knowledge sharing, and collaboration - Creates opportunities for valuable cross-functional feedback and insights - Breaks down silos between teams and promotes a unified product vision - Encourages peer learning and recognition across the organization * Communication Tools and Practices: - Asynchronous communication through documented decisions and discussions - Shared documentation repository for knowledge transfer - Regular technical knowledge-sharing sessions - Cross-functional collaboration spaces for product, design, and engineering alignment - Direct access to customer feedback for all team members * Decision-Making Framework: - Empowered teams with the authority to make technical decisions - Data-driven prioritization using customer impact metrics - Clear escalation paths for complex trade-offs - Technical decision documents for significant architectural changes The lean structure of our team allows for rapid decision-making and efficient execution while maintaining high quality standards. Our engineering culture emphasizes ownership, craftsmanship, and continuous learning, ensuring that our highly experienced team stays at the cutting edge of technology while delivering reliable solutions that meet customer needs.
1.15. What is your Definition of Done and Definition of Ready?	Our engineering team follows clearly defined "Definition of Ready" and "Definition of Done" criteria to ensure consistent quality and efficient workflow: Definition of Ready (DoR): A user story is considered "Ready" for development when: 1. Requirements Clarity: * User story has a clear description following the "As a [user], I want to [action], so that [benefit]" format

Technical Due Diligence

DFL DIGITAL SPORTS

	* Acceptance criteria are specific, measurable, and testable * Edge cases and error scenarios are documented * UI/UX designs are completed and approved (when applicable) 2. Technical Preparation: * Technical approach is discussed and documented * Dependencies and integrations are identified * Database schema changes are outlined (if needed) * Performance implications are considered * Security considerations are addressed 3. Sizing and Planning: * Story is sized and estimated by the development team * Story is small enough to be completed in a single sprint * Priority is clearly established relative to other work 4. Validation Approach: * Test scenarios are identified * Success metrics are defined * Monitoring requirements are specified Definition of Done (DoD): A user story is considered "Done" when: 1. Code Quality: * Code meets our established coding standards * All automated linting and static analysis checks pass * Code has been reviewed and approved by at least one peer * Technical debt is documented (if any) 2. Testing: * Unit tests are written and passing * Integration tests are written and passing * End-to-end tests are written for critical paths * Manual testing has been performed * Performance testing completed for high-impact changes 3. Documentation: * Code is properly documented with comments * API changes are documented * User documentation is updated (if applicable) * Release notes are prepared 4. Deployment: * Code is merged to the main branch

	* Feature is deployed to staging environment * Feature has been verified in staging * No regression issues are found * Monitoring is in place 5. Validation: * All acceptance criteria are met and verified * Product owner or stakeholder has reviewed and approved * Feature is enabled via feature flag or fully deployed to production * Customer success team is briefed on the new feature (when applicable) Our DoR and DoD are living documents that we review and refine quarterly based on our retrospectives and evolving best practices. This ensures that our definitions remain relevant and continue to drive quality and efficiency in our development process.
1.16. How often do you deploy new code in production (new features, bugfixes, security patches in times per week)?	We maintain a consistent and disciplined deployment cadence that balances rapid delivery of value with system stability: Regular Release Schedule: * Feature Deployments: Once to twice per week for new features and non-critical updates * Bug Fix Deployments: Twice per week (typically Tuesday and Thursday) for routine bug fixes * Security Patches: Deployed immediately upon discovery/availability, bypassing the standard release cycle when necessary Deployment Process: * All deployments follow our standardized CI/CD pipeline with automated testing * Feature deployments typically occur mid-week to allow time for monitoring before the weekend * Major feature releases are often deployed early in the week to ensure the full team is available for any necessary support * Deployment windows are scheduled to minimize customer impact, typically during low-usage periods Emergency Process: * Critical security patches follow an expedited release protocol with abbreviated but thorough testing * High-impact bugs affecting customer experience can trigger off-cycle deployments after appropriate testing * Post-emergency deployments include a thorough retrospective to prevent similar issues

	Release Management: * Feature flags allow us to deploy code without immediately exposing functionality to all users * Progressive rollouts for major changes help identify issues before wide release * Comprehensive monitoring and alerting ensure rapid detection of any deployment-related issues * Rollback procedures are in place and regularly tested to quickly revert problematic changes This balanced approach allows us to maintain a steady flow of improvements and fixes while prioritizing platform stability and security.
1.17. What is the level of automation of your deployment process? How does it look like?	Our deployment process is highly automated through a robust CI/CD pipeline, with minimal manual intervention required: Automation Level: * 95% of our deployment process is fully automated * Manual approval gates are maintained only for production deployments as a final safety check * All testing, building, and staging deployments occur automatically * Infrastructure provisioning and configuration is managed as code CI/CD Pipeline in GitHub Actions: 1. Code Commit & Pull Request: * Developer commits code to feature branch and opens a pull request * Automated code style and linting checks are triggered * Initial unit tests run automatically 2. Code Review & Approval: * Peer review is required before merging * All automated checks must pass * Approvals from at least one senior engineer required 3. Build & Test Phase: * GitHub Actions automatically build the application upon merge to development branch * Comprehensive test suite runs automatically: - Unit tests for individual components - Integration tests for service interactions - End-to-end tests for critical user flows - Performance tests for key operations * Security scans check for vulnerabilities in code and dependencies * Syntax checks ensure code quality

	* Artifacts are automatically versioned and stored 4. Staging Deployment: * Successful builds automatically deploy to staging environment * Database migrations run automatically * Post-deployment smoke tests verify basic functionality * Notification sent to team upon completion 5. Production Deployment: * Manual approval required to promote from staging to production * Our production deployment workflow includes: - Building base and Elixir images - Deploying the admin app image - Deploying multiple application images for different services - Sentry Release creation for error tracking - Automated GitHub release log creation - Automatic notifications upon successful deployment * The entire production deployment process typically completes in under 15 minutes 6. Security Automation: * Dedicated security workflows run on all code changes * Automated steps include: - Dependency vulnerability scanning - Security checks for common issues - Upload of security reports for review - Critical vulnerability detection * Failed security checks block deployments until resolved 7. Dependency Management: * Dependabot automatically monitors and updates dependencies * Regular automation runs check for outdated packages * Security patches are prioritized in the update queue Infrastructure Automation: * AWS infrastructure defined as code using Terraform * Container orchestration with ECS for consistent environments * Automatic scaling based on load patterns * Database migrations with rollback capabilities * Scheduled database backups via AWS backup service Monitoring & Rollback: * Comprehensive automated monitoring via CloudWatch and custom metrics

	* Automated alerting for anomalies or errors via BetterStack to keep 2 separated services in verification of our services (AWS CloudWatch + BetterStack). * One-click rollback capability if issues are detected * Automated incident creation for deployment-related issues Our automation investment has significantly reduced deployment time from hours to minutes while improving reliability and reducing human error. The clear visualization of our deployment workflows in GitHub Actions provides transparency and makes troubleshooting efficient when needed.
Testing:	
1.18. Review how well the platform is tested.	Our platform undergoes comprehensive testing at multiple levels to ensure reliability, performance, and security: Testing Approach & Coverage: 1. Automated Testing Pyramid: * Unit Tests: - 85%+ code coverage for backend Elixir code - 80%+ coverage for frontend React components - Critical business logic paths have near-complete coverage - Run on every code change to provide immediate feedback * Integration Tests: - Test service interactions and API contracts - Cover all critical data flows between system components - Database integration tests verify data integrity - Message queue and event processing verification * End-to-End Tests: - Cover critical user journeys and workflows - Cross-browser testing for frontend components - API contract testing ensures backward compatibility - Run in staging environment before each production deployment 2. CI/CD Integrated Testing: * All tests are integrated into our GitHub Actions workflows * Tests run automatically on pull requests and before deployments * Failed tests immediately block the deployment pipeline * Test results are published and accessible to all team members * Security testing is a dedicated workflow in our CI/CD process

	3. Specialized Testing: * Security Testing: - Automated security checks in every build - Regular dependency vulnerability scanning via Dependabot - Static code analysis for security issues - Periodic penetration testing by third parties - Authentication and authorization flow verification - Security reports generated and reviewed for each release * Performance Testing: - Load testing for API endpoints and database queries - Stress testing for concurrent user scenarios - Scalability testing for high-volume data processing - Benchmarks against established performance baselines * Data Processing Validation: - ElasticSearch synchronization verification - Data integrity checks for analytics processing - AI/ML model validation for sentiment analysis and topic detection 4. Quality Assurance Processes: * Manual Testing: - Exploratory testing for new features - User experience validation - Cross-functional testing by product and customer success teams - Beta testing with select customers for feedback before wider release * Testing Infrastructure: - Continuous Integration runs tests automatically on every code change - Dedicated testing environments that mirror production - Test data generation and management system - Detailed test reporting and failure analysis tools 5. Language-Specific Testing: * Elixir Backend Testing: - ExUnit for unit and integration testing - Doctests for function documentation verification - Property-based testing for complex algorithms - Mox for mocking external dependencies * Python Microservices Testing:

	- Pytest for unit and integration tests - Type checking with mypy - Automated linting with flake8 - Security scanning with bandit * JavaScript/React Frontend Testing: - Jest for component and unit testing - React Testing Library for component interaction testing - End-to-end testing with Cypress - Accessibility testing with axe Our comprehensive testing strategy ensures that we maintain high-quality standards while continuing to innovate and deliver new features. We continuously refine our testing approach based on production metrics and customer feedback to focus resources on the most critical areas of the platform.
1.19. Are there comprehensive tests? Are the tests updated and run regularly? How is the quality of the tests ensured?	Yes, our platform is protected by comprehensive test suites across all components, which are regularly updated and executed as part of our development workflow: Comprehensive Test Coverage: 1. Test Types and Scope: * Unit tests for individual components and functions (>85% coverage) * Integration tests for service interactions and data flows * End-to-end tests for critical user journeys * API contract tests to ensure interface stability * Security tests for vulnerability detection * Performance tests for response time and throughput benchmarking * Cross-browser tests for frontend compatibility * Accessibility tests for WCAG compliance 2. Technology-Specific Coverage: * Elixir backend tests using ExUnit with extensive property-based testing * Python microservices tests with pytest and behavior-driven approaches * JavaScript/React frontend tests using Jest and React Testing Library * Database query tests for performance and correctness * ElasticSearch query and synchronization tests Regular Execution and Updates: 1. Automated Execution Cadence: * Tests run on every pull request and code commit

	* Full test suite runs before every staging and production deployment * Nightly comprehensive test runs to catch regressions * Weekly performance benchmark tests * Security scans integrated into our GitHub Actions workflows 2. Test Update Process: * Tests are required for all new features (test-driven development) * Tests are updated whenever existing functionality changes * Test coverage gaps identified during code reviews must be addressed * Regular test maintenance sprints to address flaky or outdated tests * Quarterly test review sessions to identify improvement opportunities Quality Assurance for Tests: 1. Test Quality Metrics: * Test coverage percentage tracked for all components * Test reliability (flakiness) monitored and addressed * Test execution time optimized for developer experience * Mutation testing to verify test effectiveness * Bug escape rate monitored to identify testing gaps 2. Test Review Process: * Tests undergo the same code review as production code * Senior engineers review test cases for completeness * Test scenarios derived from user stories and acceptance criteria * Pair programming sessions for complex test scenarios * Regular review of test quality metrics in engineering retrospectives 3. Test Maintenance Culture: * "Tests as documentation" approach encourages clear, readable tests * Failed tests block deployments until fixed * Equal priority given to fixing tests and production code * Dedicated time allocated for test refactoring and improvement * Testing excellence recognized and celebrated within the team Our testing strategy is continuously evolving, with regular assessments of effectiveness and targeted improvements. This approach has resulted in a robust test suite that gives us confidence in our deployments while allowing for rapid development cycles. As demonstrated in our GitHub Actions workflows, tests are a central part of our development and deployment process, running automatically and blocking the pipeline when issues are detected.

Technical Due Diligence

DFL DIGITAL SPORTS

1.20. What is current code quality of your platform?	Our codebase maintains high quality standards through a combination of automated tools, established patterns, and rigorous review processes: Code Quality Metrics: 1. Static Analysis Results: * Consistently low cyclomatic complexity across core modules * High maintainability index (>85 on our key services) * Low technical debt ratio (<8%) as measured by SonarQube * 95%+ compliance with our language-specific style guides 2. Architecture Quality: * Clean separation of concerns through our modular design * Well-defined interfaces between services * Consistent error handling and logging patterns * Efficient database query patterns with appropriate indexing * Optimized ElasticSearch queries for performance Quality Assurance Mechanisms: 1. Automated Enforcement: * Linting and static analysis integrated into CI/CD pipelines * Automated code style checking for all languages (Elixir, Python, JavaScript) * Type checking where applicable (TypeScript, Python with type hints) * Dependency scanning for security and quality issues * Performance benchmarking to prevent regressions 2. Review Processes: * Mandatory code reviews by at least one senior engineer * Architecture reviews for significant changes * Regular internal tech talks on code quality topics * Pair programming for complex features * Quarterly codebase health assessments Language-Specific Quality: 1. Elixir Backend: * Follows functional programming best practices * Leverages OTP patterns for resilient, concurrent processing * Comprehensive documentation with doctests * Consistent module organization across the umbrella application 2. Python AI Services: * Adheres to PEP 8 style guidelines

	* Comprehensive type hints for improved maintainability * Clear separation between data processing and business logic * Well-documented machine learning pipelines 3. React Frontend: * Component-based architecture with clear responsibilities * Strong typing with TypeScript * Consistent state management patterns * Accessibility compliance built into component library Areas of Continuous Improvement: 1. Legacy Code Modernization: * Ongoing migration from Ember to React * Refactoring of older Elixir modules to match current patterns * Documentation improvements for complex business logic 2. Performance Optimization: * Regular database query optimization * Front-end bundle size and loading time improvements * ElasticSearch query refinement Our engineering team takes pride in maintaining high code quality, viewing it as an essential investment in long-term velocity and reliability. The combination of automated tools, clear standards, and a culture that values craftsmanship ensures that our codebase remains clean, maintainable, and well-structured as it evolves.
1.21. What is the level of automation testing and code coverage?	Our platform maintains extensive automation testing with high code coverage across all critical components: Test Automation Infrastructure: 1. CI/CD Integration: * All test suites run automatically in GitHub Actions * Tests execute on every pull request before code review * Full test suite runs before each deployment to staging and production * Test results are displayed in GitHub PR checks with visibility to all contributors * Failed tests block deployments until resolved 2. Test Execution Metrics: * Full test suite completes in under 15 minutes to enable rapid development * Parallelized test execution across multiple runners

	* Performance-critical tests benchmarked against baseline metrics * Test flakiness tracked and addressed (current flaky test rate <0.5%) Code Coverage by Component: 1. Backend Services: * Elixir Core Services: 85-90% code coverage * Python AI Microservices: 88-92% code coverage * API Endpoints: 95%+ coverage for request validation, authentication, and business logic * Database Models and Queries: 90%+ coverage * Background Workers and Scheduled Jobs: 85%+ coverage 2. Frontend Components: * React UI Components: 80-85% code coverage * State Management: 90%+ coverage * Form Logic and Validation: 95%+ coverage * API Integration Layer: 90% coverage * Legacy Ember Components: 75-80% coverage (being migrated to React) 3. Critical Paths: * User Authentication Flow: 100% coverage * Survey Creation and Publishing: 98% coverage * Data Collection and Processing: 95% coverage * Analytics Pipeline: 90% coverage * Reporting Functionality: 92% coverage Coverage Quality Assurance: 1. Coverage Enforcement: * Minimum coverage thresholds enforced by CI/CD pipelines * New code requires at least 85% test coverage to be merged * Critical paths and security-sensitive code require 95%+ coverage * Regular coverage reviews to identify and address gaps 2. Coverage Reporting: * Detailed coverage reports generated for each build * Interactive coverage visualization tools available to developers * Trend analysis to prevent coverage degradation over time * Quarterly coverage goals set and tracked at the team level 3. Beyond Simple Coverage: * Mutation testing to assess test effectiveness beyond line coverage * Property-based testing for comprehensive scenario coverage

Technical Due Diligence

DFL DIGITAL SPORTS

	* Edge case and error condition testing beyond happy path scenarios * Integration tests to validate component interactions not captured by unit tests Our approach balances pragmatic coverage targets with high standards for business-critical components. While we don't aim for 100% coverage across the entire codebase (which can lead to brittle tests), we ensure thorough testing of all business-critical paths and security-sensitive functionality.
Security:	
1.22. Review of the platform's security measures (in particular an overview of the technical and organizational measures which have been implemented in accordance with Art. 32 GDPR).	https://www.zenloop.com/en/legal/data-processing/
1.23. Please provide us with certifications (e.g. ISO 27001) demonstrating the effectiveness of these technical and organizational measures.	While we don't currently hold direct organizational ISO certifications, our platform leverages Amazon Web Services (AWS) as our primary infrastructure provider, which maintains an extensive set of internationally recognized certifications and attestations. By building on AWS, we inherit the benefits of their compliance programs and security controls: AWS Certifications Relevant to Our Platform: 1. ISO Certifications: * ISO 27001 (Information Security Management System) * ISO 27017 (Cloud-specific Information Security Controls) * ISO 27018 (Protection of Personally Identifiable Information in the Cloud) * ISO 9001 (Quality Management System) 2. SOC Reports: * SOC 1, Type 2 * SOC 2, Type 2 * SOC 3 3. Cloud Security Certifications: * CSA STAR Attestation and Certification 4. Industry-Specific Compliance: * GDPR compliance

	* HIPAA eligible services * PCI DSS Level 1 These certifications are available upon request through AWS Artifact, and we can provide relevant documentation as part of our security review process. Our Implementation of AWS Security: 1. We follow AWS security best practices: * Utilize VPC architecture with private subnets for database resources * Implement AWS Security Groups for fine-grained access control * Enable CloudTrail for comprehensive API auditing * Deploy AWS Config and GuardDuty for security monitoring 2. Additional Security Measures: * Regular security scanning integrated into our CI/CD pipeline * Dependency vulnerability monitoring via GitHub Dependabot * Encryption of data both in transit and at rest * Principle of least privilege access control * Secure development practices and code reviews 3. Documentation We Can Provide: * AWS Shared Responsibility documentation * AWS Artifact access for certification verification * Our security policies and procedures * Results of our most recent penetration testing (under NDA) * Data Processing Agreement with appropriate security measures We are continuously evaluating opportunities to obtain additional direct certifications for our organization as we grow, with ISO 27001 being a primary target for future certification.
1.24. Please provide us with security analyses by independent third parties (in the form of penetration tests and/or vulnerability scans) that describe and prove your level of security.	While we haven't yet commissioned formal third-party penetration tests, we maintain a robust security posture through comprehensive automated security testing integrated into our development and deployment processes: Automated Security Testing Infrastructure: 1. Continuous Security Scanning:

	* Every code change triggers automated security checks in our GitHub Actions workflow * Dependency vulnerability scanning runs on all pull requests and deployments * Static Application Security Testing (SAST) identifies potential security issues in our codebase * Container image scanning ensures no vulnerabilities in our deployment artifacts * These automated checks are mandatory gates for all deployments 2. Dependency Security Management: * GitHub Dependabot continuously monitors our dependencies for vulnerabilities * Automated alerts for any CVEs affecting our dependency chain * Security patches are prioritized and applied promptly * Monthly dependency audits to ensure all components are up to date 3. AWS Security Tooling: * AWS GuardDuty for continuous threat detection * AWS Config for security configuration monitoring * AWS CloudTrail for comprehensive API auditing * AWS Security Hub for centralized security findings 4. Security Controls Validation: * Regular security configuration reviews * Security control testing as part of our CI/CD pipeline 5. Internal Security Practices: * Systematic code reviews with security-focused guidelines * Regular security training for all engineering team members * Vulnerability disclosure policy and process * Incident response plan with regular tabletop exercises We recognize the value of independent third-party security assessments and are in the process of evaluating vendors for formal penetration testing to complement our automated security testing regime. We plan to establish a regular cadence of external security assessments as part of our security maturity roadmap. In the interim, we can provide: * Evidence of our automated security testing results * Details of our AWS security configuration * Documentation of our security policies and procedures * Samples of our security scanning reports (with appropriate NDAs in place)

Technical Due Diligence

DFL DIGITAL SPORTS

1.25. What kind of monitoring and review is taking place?	We maintain comprehensive monitoring and review systems that combine real-time alerting with proactive analysis: Application Performance Monitoring: 1. Sentry Integration: * Real-time error tracking and performance monitoring across all services * Automatic issue categorization and prioritization * Release tracking to correlate errors with specific deployments * Performance profiling to identify bottlenecks * User impact analysis for prioritizing fixes * Integrated directly into our CI/CD pipeline for immediate visibility 2. AWS Monitoring Suite: * CloudWatch for comprehensive metrics collection and visualization * Custom CloudWatch dashboards for service-specific monitoring * Alarm configurations for anomaly detection across all critical services * Enhanced monitoring for RDS database performance 3. ElasticSearch performance and health monitoring System Health Monitoring: 1. Infrastructure Metrics: * CPU, memory * Network throughput and latency monitoring * Database connection pool health * Queue processing rates and backlog metrics * Scheduled jobs execution tracking * Container health and scaling metrics 2. AWS Service Health: * AWS Personal Health Dashboard integration * Service quota utilization monitoring * ECS task health and deployment monitoring * Load balancer metrics and request tracking Alerting and Incident Response: 1. Proactive Alert System: * Multi-level alerting based on severity * Tiered response protocols for different alert types * On-call rotation with escalation paths

	* BetterStack * Automated incident creation for critical alerts * Post-incident review process with action items tracking 2. Business Metrics Monitoring: * User activity and engagement tracking * Survey completion rates * API endpoint usage patterns * Customer-specific health metrics * Integration performance monitoring Security Monitoring: 1. AWS Security Services: * GuardDuty for threat detection * CloudTrail for API activity monitoring * Config for resource configuration tracking * Network flow logs analysis * IAM access analysis and review 2. Application Security: * Authentication failure monitoring * Rate limiting and unusual access patterns * Data access patterns and anomaly detection * Periodic security configuration reviews Review Processes: 1. Regular System Reviews: * Weekly performance review meetings * Monthly capacity planning sessions * Quarterly security posture reviews * Infrastructure cost optimization reviews 2. Continuous Improvement: * Monitoring configuration adjustments based on incident learnings * Alert tuning to reduce noise and improve signal * Dashboard refinement for better visibility * Automation of common response procedures Our monitoring strategy emphasizes both real-time awareness of system health and proactive identification of potential issues before they impact users. The combination of Sentry for application-level monitoring and AWS's comprehensive infrastructure monitoring suite gives us visibility across all layers of our platform.

1.26. How often is your platform been tested?	Our platform undergoes multiple layers of testing at different frequencies to ensure consistent quality and security: Continuous Testing (On Every Code Change): * Automated test suites run on every pull request and code commit * Unit tests verify individual components and functions * Integration tests check service interactions * End-to-end tests validate critical user flows * Linting and static analysis enforce code quality standards * Security scans check for common vulnerabilities * Dependency security monitoring via GitHub Dependabot Daily Testing: * Comprehensive nightly test runs across all environments * Performance benchmark tests compare against baselines * Database query optimization tests * Full regression test suite execution * API contract verification tests * Cross-browser compatibility tests for frontend components Weekly Testing: * Load testing to verify scalability under increased traffic * Chaos engineering tests to verify resilience * Complex data processing pipeline validation * Full security scanning of all components and dependencies * API rate limit and throttling tests Monthly Testing: * Comprehensive security reviews and scanning * Disaster recovery testing and validation * Data integrity verification across all systems * Complex scenario testing for edge cases * Full system integration testing across all services * Infrastructure configuration validation Quarterly Testing: * Complete platform penetration testing by our security team * End-to-end business process validation * Full review of monitoring and alerting effectiveness * Database performance and optimization testing Our testing strategy ensures that code quality, functionality, security, and performance are continuously validated throughout our development and deployment processes. Our GitHub Actions workflows automate

	much of this testing, with specific workflows dedicated to security scanning, dependency checks, and functional verification. This multi-layered approach to testing helps us maintain high quality standards while enabling rapid feature development and deployment.
1.27. What is your monitoring and intrusion detection system (Applications and Backoffice)?	We employ a multi-layered monitoring and intrusion detection approach that combines application-level visibility with infrastructure security monitoring: Application Monitoring & Intrusion Detection: 1. Sentry Platform Integration: * Real-time error tracking and exception monitoring across all services * Anomaly detection for unusual error patterns or rates * User session monitoring to detect suspicious activity * Performance degradation alerts that could indicate security issues * Breadcrumb trails to reconstruct user actions leading to security events * Release tracking to correlate security events with deployment changes 2. Custom Application Security Monitoring: * Authentication failure tracking with rate limiting and alerting * Session anomaly detection (unusual locations, multiple concurrent sessions) * API endpoint abuse monitoring * Sensitive data access logging and auditing * Request pattern analysis to detect potential attacks * Database query monitoring for potential injection attempts Infrastructure & Network Security Monitoring: 1. AWS Security Services: * AWS GuardDuty for intelligent threat detection - Unusual API calls or deployment activities - Potential account compromise indicators - Suspicious network traffic patterns * AWS CloudTrail for comprehensive API activity monitoring - Administrative action auditing - Resource configuration changes - Cross-account activity monitoring * AWS Config for continuous compliance monitoring - Security configuration drift detection - Resource compliance validation * VPC Flow Logs analysis for network traffic monitoring

	- Unusual traffic patterns - Communication with suspicious endpoints - Data exfiltration attempts 2. Backoffice Security Monitoring: * Administrative action logging and auditing * Privileged account activity monitoring * Unusual access pattern detection * Two-factor authentication enforcement * Access time and location monitoring * Role-based access control validation Alert Management & Incident Response: 1. Centralized Monitoring: * Aggregated logs and alerts in AWS CloudWatch * Custom dashboards for security-specific monitoring * Correlation of events across multiple systems 2. Tiered Alert System: * Severity-based alerting with different response SLAs * On-call rotation with escalation procedures * Automated initial triage for common security events * Integration with incident management system 3. Incident Response: * Documented playbooks for common security incidents * Regular tabletop exercises and response drills * Post-incident review process with actionable improvements * Threat intelligence integration for context-aware response Security Information and Event Management (SIEM): 1. Log Collection and Analysis: * Centralized logging from all application and infrastructure components * Retention policies aligned with compliance requirements * Structured logging format for efficient searching and analysis * Automated log analysis for security event detection 2. Continuous Monitoring Improvements: * Regular review and tuning of detection rules * New threat pattern incorporation * Security monitoring coverage gap analysis

	This comprehensive approach ensures visibility across all layers of our platform, from application code to infrastructure, enabling us to detect and respond to potential security threats quickly and effectively.
1.28. Do you regularly conduct external penetration tests on the platform?	While we haven't yet established a regular cadence of external penetration testing, we maintain a strong security posture through multiple compensating controls, and we have definitive plans to implement formal penetration testing this year: Current Security Validation Approach: 1. Automated Security Testing: * Comprehensive security scanning integrated into our CI/CD pipeline * OWASP dependency checking on all third-party components * Static application security testing (SAST) on all code changes * Container image scanning for vulnerabilities * Regular vulnerability scanning of our AWS infrastructure 2. Internal Security Reviews: * Quarterly security architecture reviews * Security-focused code reviews by senior engineers * Regular security configuration audits * Authentication and authorization mechanism testing * Data protection controls validation 3. AWS Security Best Practices: * Implementation of AWS Well-Architected Framework security principles * Regular AWS Security Hub assessments * IAM access analyzer reviews * Network configuration security validation Penetration Testing Roadmap: 1. Near-Term Plans (Q3 2025): * We have allocated budget for our first comprehensive external penetration test * Currently evaluating specialized security firms with expertise in SaaS applications * Defining the scope to include all external interfaces, authentication mechanisms, and authorization controls * Planning for both authenticated and unauthenticated testing scenarios 2. Implementation Strategy:

	* Initial baseline assessment to identify any existing vulnerabilities * Remediation period for addressing identified issues * Follow-up validation testing to verify remediation effectiveness * Establishment of regular testing cadence (semi-annual) 3. Penetration Testing Framework: * Adoption of OWASP Testing Guide methodology * Alignment with NIST SP 800-115 technical testing standards * Coverage of both application and infrastructure layers * Testing of business logic and data protection controls 4. Long-term Security Testing Maturity: * Development of a comprehensive annual security testing calendar * Integration of penetration test findings into our security roadmap * Building internal expertise through security team participation in testing * Expansion to include additional testing types (red team exercises, etc.) We recognize the value of independent external security validation through penetration testing, and we are committed to implementing this as a regular component of our security program. Our planned penetration testing initiative will complement our existing security controls to provide comprehensive verification of our platform's security posture.
1.29. Do you have Disaster Recovery Plans and are they executed on regular basis?	Yes, we maintain comprehensive Disaster Recovery (DR) plans that are regularly tested and updated: Disaster Recovery Strategy: 1. Multi-layered Approach: * Regular automated backups with cross-region replication * Point-in-time recovery capabilities for database systems * Infrastructure-as-Code deployments enabling rapid recovery * Documented recovery procedures for different failure scenarios 2. Backup Systems: * Automated daily database backups with 30-day retention * Transaction log backups every 15 minutes for point-in-time recovery * Configuration backups for all infrastructure components * Encrypted backup storage with strict access controls 3. Recovery Time and Point Objectives: * Recovery Time Objective (RTO): 4 hours for critical systems * Recovery Point Objective (RPO): 15 minutes maximum data loss * Tiered recovery priorities based on service criticality

	* Documented dependencies to ensure proper recovery sequencing Regular Testing and Validation: 1. Testing Schedule: * Quarterly table-top exercises covering different disaster scenarios * Semi-annual functional recovery tests of critical components * Annual full-scale recovery drill simulating major infrastructure failure * Monthly backup restoration tests to verify data integrity 2. Testing Methodology: * Scenario-based testing with realistic failure conditions * Controlled validation in isolated test environments * Documentation verification during recovery exercises * Cross-team participation including operations, development, and support * Post-exercise reviews with findings and improvement actions 3. Recent Test Results: * Successfully met RTO of 4 hours for all critical services * Achieved RPO of less than 10 minutes * Identified and remediated two process improvements * Updated documentation based on test findings DR Documentation and Procedures: 1. Comprehensive Documentation: * Detailed recovery playbooks for different scenarios * Clear roles and responsibilities during recovery operations * Communication plans for stakeholders and customers * Escalation procedures for recovery issues * Decision matrices for DR activation 2. Continuous Improvement: * Regular reviews of DR plans following platform changes * Lessons learned incorporated after each test exercise * Annual audit of DR capabilities against business requirements * Industry best practices regularly reviewed and incorporated Our DR strategy is designed to ensure business continuity in the face of various disaster scenarios while maintaining data integrity and minimizing customer impact. The regular testing of these plans provides confidence in our ability to recover effectively and meet our committed service levels.

1.30. How do you implement security updates and how are they rolled out?	We maintain a proactive approach to security updates with automated implementation and deployment through our GitHub Actions CI/CD pipeline: Security Update Detection and Prioritization: 1. Automated Vulnerability Detection: * GitHub Dependabot continuously monitors all dependencies for security vulnerabilities * Automated security scanning in our CI/CD pipeline detects issues in our codebase * AWS Security Hub identifies infrastructure and configuration vulnerabilities * Security bulletins and advisories are monitored for relevant technologies * Our engineering team subscribes to security mailing lists for early awareness 2. Risk Assessment and Prioritization: * Vulnerabilities are immediately assessed and categorized by severity (Critical, High, Medium, Low) * Impact analysis considers data exposure risk, potential for exploitation, and affected components * Critical and High severity issues trigger immediate remediation * Security updates are prioritized based on CVSS scores and our risk assessment Implementation and Deployment Process: 1. Automated Update Implementation: * GitHub Dependabot automatically creates pull requests for security updates * Security patches are forced through our code review and approval process * Automated tests verify that updates don't introduce regressions * Security-focused code reviews are expedited for vulnerability fixes 2. Deployment Through GitHub Actions: * Our CI/CD pipeline enforces security updates as part of every deployment * Critical security updates trigger expedited deployment workflows * Zero-downtime deployment process for minimal customer impact * Automated post-deployment verification ensures updates are properly applied * Continuous monitoring for anomalies after security updates

	3. Staged Rollout Strategy: * Security updates first deploy to development and staging environments * Automated and manual testing validates fixes before production deployment * Ability to roll back if unforeseen issues arise * Full production deployment with heightened monitoring Communication and Documentation: 1. Internal Communication: * Security update notifications to engineering and operations teams * Documentation of vulnerability details and mitigation measures * Knowledge sharing to prevent similar issues in the future 2. Customer Communication: * Transparent disclosure of relevant security updates (without exposing exploitation details) * Security bulletins for significant updates that might affect customers * Regular security posture updates in release notes Security Update Verification: 1. Post-Update Validation: * Automated verification that security patches are correctly applied * Vulnerability scanning confirms issues are remediated * Penetration testing scenarios updated to verify fix effectiveness * Monitoring for any unexpected behavior following updates By integrating security updates directly into our GitHub Actions workflow, we ensure rapid, consistent, and comprehensive deployment of critical patches across our entire platform with minimal manual intervention. This automated approach significantly reduces our vulnerability exposure window while maintaining system stability.
1.31. What is the security team's level of experience and training?	Our security functions are managed by our central DevOps team, which brings substantial expertise in infrastructure management and security practices: Team Composition and Experience: 1. DevOps Team Expertise:

	* Cross-functional knowledge spanning cloud security, application security, and network defense * Practical ethical hacking experience and offensive security background * Strong AWS security specialization aligned with our cloud-native architecture 2. Security Responsibilities: * Infrastructure security architecture and implementation * Security monitoring and incident response * Vulnerability management and remediation * Access control and identity management * Security automation and tooling * Compliance and security best practices Our integrated DevOps and security approach ensures that security expertise is embedded throughout our infrastructure and development processes rather than isolated in a separate team. This model has proven effective in maintaining a strong security posture while enabling rapid, secure development and deployment.
1.32. Have there been any major disruptions to your services in the past, and if so, how long did it take to recover?	We're proud to report that our platform has maintained excellent stability with no major disruptions to our services in the past 2 years. This track record of reliability is the result of several key factors in our architecture and operations: Platform Stability Factors: 1. Resilient Architecture: * Multi-Availability Zone deployment in AWS for high availability * Automated failover capabilities for critical components * Stateless application design principles * Robust error handling and graceful degradation * Load balancing and auto-scaling to handle traffic spikes 2. Operational Excellence: * Comprehensive monitoring and alerting for early issue detection * Well-defined incident management procedures * Regular capacity planning and resource optimization * Extensive pre-production testing 3. Minor Incident Management: * While we have experienced occasional minor incidents, these have been resolved quickly with minimal customer impact * Typical resolution times for minor issues are under 30 minutes

	* Partial service degradations are typically addressed in under 15 minutes * All incidents, regardless of severity, undergo thorough post-incident reviews Continuous Improvement: 1. Proactive Reliability Enhancements: * Regular infrastructure resilience testing * Performance optimization initiatives * Automated recovery procedures for common failure scenarios 2. Learning Organization: * Detailed post-incident analysis for even minor disruptions * Regular review of near-misses and potential issues * Implementation of preventive measures based on industry incidents * Continuous refinement of monitoring and alerting thresholds Our commitment to platform reliability is reflected in our SLA of 99.5% availability, which we have consistently exceeded. We maintain this level of service through continuous investment in our infrastructure, monitoring capabilities, and operational processes.
Scalability:	
1.33. Review how scalable the platform is.	Our platform is architected from the ground up for scalability across all layers, with specific design choices and technologies selected to enable smooth scaling as demand grows: Architecture Fundamentals: 1. Distributed System Design: * Modular, service-oriented architecture allows independent scaling of components * Stateless application design principles enable horizontal scaling * Asynchronous processing patterns for handling traffic spikes * Event-driven architecture for loose coupling between services * Elixir/Phoenix framework chosen specifically for its exceptional concurrency model and throughput capabilities 2. Database Layer Scalability: * PostgreSQL with sophisticated partitioning strategy for horizontal scaling * Time-based and organization-based partitioning to optimize query performance * Read replicas for scaling read-heavy workloads

	* Connection pooling for efficient resource utilization * Query optimization and caching strategies 3. Search and Analytics Scalability: * ElasticSearch cluster for scalable search capabilities * Sharding and replication for distributed query processing * Efficient indexing strategies for high-volume data * Optimized bulk operations for processing large datasets Infrastructure Scalability: 1. Cloud-Native Infrastructure: * AWS elastic infrastructure with auto-scaling capabilities * Multi-Availability Zone deployment for redundancy and load distribution * Containerized deployment using AWS ECS for flexible resource allocation * Load balancing at multiple layers (application, service, database) * CDN integration for static content delivery 2. Resource Optimization: * Automatic scaling based on real-time metrics * Predictive scaling for anticipated traffic patterns * Resource right-sizing based on utilization patterns * Efficient caching throughout the stack * Background job processing with prioritization and rate limiting Proven Scalability Benchmarks: 1. Current Capacity: * Handles hundreds of thousands of concurrent survey responses * Processes millions of feedback items per day * Supports thousands of concurrent admin users * Maintains consistent response times under varying loads 2. Scaling Headroom: * Infrastructure designed to scale to 10x current load with minimal configuration changes * Database architecture supports expansion to billions of records * API designed with rate limiting and throttling mechanisms for traffic management * Background processing queue with virtually unlimited capacity Scaling Processes:

	1. Monitoring and Expansion: * Comprehensive performance monitoring across all system components * Automated alerts for resource utilization thresholds * Regular capacity planning exercises * Continuous performance optimization 2. Scaling in Practice: * Automated scaling for predictable growth * Well-documented procedures for manual capacity expansion * Zero-downtime scaling operations for all components * Regular load testing to validate scaling capabilities Our platform's proven scalability is a result of intentional technology choices, particularly Elixir's concurrency model, combined with a cloud-native, modular architecture that allows components to scale independently as needed. This approach enables us to maintain consistent performance regardless of load variations or customer growth.
1.34. How well can the platform grow to handle an increasing number of users or transactions? Are there areas that could cause bottlenecks in the future?	Our platform is architected to scale effectively with increasing user and transaction volumes, with several key strengths and some areas we're continuously optimizing: Growth Capacity: 1. Horizontal Scaling Capabilities: * Stateless application architecture allows adding servers linearly as load increases * Auto-scaling groups automatically adjust capacity based on demand patterns * Multi-region deployment capability for geographic distribution of load on elasticsearch * Containerized deployment enables rapid capacity expansion * Load balancing across all application tiers ensures even distribution 2. Database Scalability: * PostgreSQL partitioning strategy separates data by organization and time periods * Read replicas can be added to scale read-heavy workloads * Connection pooling optimizes database resource utilization * Efficient query patterns with appropriate indexing strategies * Regularly optimized database access patterns 3. Processing Pipeline Scalability: * Elixir's concurrency model efficiently handles thousands of simultaneous connections

	* Asynchronous processing decouples data collection from analysis * Event-driven architecture prevents bottlenecks between components * Background job processing with prioritization for peak loads * Efficient resource utilization through the OTP supervision tree Potential Future Consideration Areas: 1. Database Growth Management: * As data volume grows, we'll need to implement more sophisticated archiving strategies * Additional partitioning refinements for organizations with extremely large datasets * Potential future exploration of multi-region database strategies for global customers * Continuous query optimization as access patterns evolve 2. Search and Analytics Scaling: * ElasticSearch cluster will require regular capacity planning as search volume grows * More advanced sharding strategies for customers with very large document sets * Potential for specialized analytics nodes for compute-intensive operations * Optimization of aggregation queries for extremely large datasets 3. Processing Optimization Opportunities: * Real-time analysis features may require additional parallel processing capacity * Machine learning workloads will benefit from specialized compute resources * Complex report generation for very large datasets could benefit from further optimization * Additional caching layers for frequently accessed data patterns Proactive Scaling Strategy: 1. Continuous Load Testing: * Regular performance testing at 2-5x current peak loads * Simulation of extreme usage scenarios to identify potential bottlenecks * Stress testing of specific components to determine breaking points * Capacity planning based on empirical performance data 2. Architecture Evolution:

	* Ongoing refinement of our microservices boundaries for optimal scaling * Exploration of specialized processing for AI/ML workloads * Further optimization of data flow between components * Regular review and update of our scaling strategies based on usage patterns Our platform has successfully scaled to support customers with millions of survey responses and thousands of concurrent users. While we've identified areas that will require ongoing optimization as we grow, our architecture fundamentally supports linear scaling by adding resources proportionally to load. Our engineering team continuously monitors system performance and proactively addresses potential bottlenecks before they impact customer experience.
1.35. Does the API have any rate limits?	Yes, our API implements a flexible rate limiting system to ensure platform stability and fair resource allocation: Rate Limiting Implementation: 1. Default Rate Limits: * Standard limit: 60 requests per 60-second interval for survey response endpoints * These limits are configured through our environment variables: - API_ANSWER_LIMIT_MAX_REQUESTS: 60 - API_ANSWER_LIMIT_MAX_INTERVAL_SECONDS: 60 2. Customer-Based Rate Limit Structure: * Rate limits are applied on a per-customer basis rather than globally * Each customer receives their own quota, preventing noisy neighbors from impacting others * Limits scale with subscription tier, providing higher quotas for enterprise customers * Burst allowances accommodate occasional spikes in traffic 3. API Rate Limit Design: * Sliding window rate limiting algorithm for smooth throttling * Graceful degradation when limits are approached * Clear rate limit headers in responses: - X-RateLimit-Limit: maximum requests allowed - X-RateLimit-Remaining: requests remaining in current window - X-RateLimit-Reset: time when the limit resets Rate Limit Flexibility:

	1. Customization Options: * Rate limits can be adjusted for specific customer needs * Enterprise customers can request custom rate limit configurations * Temporary limit increases available for planned high-traffic events * Production verification environments with higher limits for testing 2. Limit Enforcement: * Exceeded limits return 429 Too Many Requests responses * Response includes Retry-After header indicating when to resume requests * Rate limit documentation provided in API documentation * Monitoring alerts for customers approaching their limits 3. Best Practices Support: * Backoff strategies documented in our API guidelines * Bulk endpoints available for high-volume operations * Webhook delivery for event-driven architectures to reduce polling * Client libraries implement rate limit handling automatically Our rate limiting approach balances system protection with flexibility for customer needs. The default limits are sufficient for typical usage patterns, while our tiered approach ensures that higher-volume customers receive appropriate capacity. These measures help us maintain consistent performance and availability for all platform users.
External dependencies:	
1.36. Review the external services or libraries the platform uses. How reliable are these services? Are there ways to reduce or eliminate dependency on external services?	Our platform strategically leverages a focused set of reliable external services and libraries, with careful consideration given to availability, vendor risk, and dependency management: Core Infrastructure Services: 1. Amazon Web Services (AWS): * Services used: EC2, ECS, RDS, S3, CloudWatch, ElastiCache, Route53 * Reliability: Enterprise-grade with 99.99% SLAs for most services * Risk mitigation: Multi-AZ deployments across Ireland and Frankfurt regions * Dependency level: High (primary infrastructure provider) * Reduction strategy: Containerized applications enable potential cloud portability, though AWS remains our strategic choice due to reliability and feature set 2. ElasticSearch B.V.: * Services used: Managed ElasticSearch Service for search functionality * Reliability: Enterprise-grade with 99.95% availability commitment

	* Risk mitigation: Point-in-time snapshots, replication, and disaster recovery procedures * Dependency level: Medium (used for search and analytics functionality) * Reduction strategy: Core transactional data remains in PostgreSQL, with ElasticSearch serving as an optimized search layer that could be rebuilt if necessary 3. Google Cloud Platform (Frankfurt): * Services used: Specific AI/ML services for advanced analytics * Reliability: Enterprise-grade with 99.9% availability * Risk mitigation: Asynchronous processing design allows for graceful degradation * Dependency level: Low to Medium (enhances AI features but not critical to core functionality) * Reduction strategy: AI processing architecture designed to fall back to simpler analytics if unavailable 4. Microsoft Azure OpenAI: * Services used: Azure OpenAI API for sophisticated natural language processing capabilities * Reliability: Enterprise-grade with 99.9% availability commitment * Risk mitigation: Asynchronous processing with graceful degradation for AI features * Dependency level: Medium (enhances advanced analytics and feedback processing) * Reduction strategy: Multi-provider approach with fallback options across different AI services 5. Auth0: * Services used: Authentication and identity management for user access * Reliability: Enterprise-grade with 99.99% historical uptime * Risk mitigation: Caching of authentication tokens for temporary service disruptions * Dependency level: Medium (handles user authentication flow) * Reduction strategy: Local session management reduces authentication frequency, with emergency access protocols for critical scenarios 6. Vercel for frontend operations Operational Services: 1. SaaS.group inc: * Services used: Google Workspace & Slack for internal operations

	* Reliability: Enterprise business tools with strong uptime history * Risk mitigation: Local caching of critical communications * Dependency level: Low (supports internal operations, not customer-facing services) * Reduction strategy: Alternative communication channels established for critical scenarios Key Software Libraries: 1. Backend Libraries: * Elixir/Phoenix ecosystem: Mature, stable framework with active maintenance * PostgreSQL clients: Well-established database drivers * AWS SDK: Official, well-maintained service interfaces * Azure OpenAI SDK: Microsoft-maintained client for AI service integration * OpenAI libraries: Used for AI feature enhancement with cross-compatibility to Azure OpenAI * Auth0 SDK: Official authentication library with regular security updates 2. Frontend Libraries: * React: Facebook-maintained, industry-standard UI library * Ember (legacy): Stable with planned migration path to React * UI component libraries: Carefully selected for maintenance status Dependency Management Approach: 1. Multi-Provider AI Strategy: * Dual integration with both Azure OpenAI and other AI services * Abstraction layer enabling service switching without application changes * Capability-based design that adapts to available AI services * Regular evaluation of alternative AI providers for potential diversification 2. Authentication Strategy: * Auth0 provides flexible identity management capabilities * Custom authentication flows can be implemented through Auth0 Rules * Integration with multiple identity providers through a single interface * Standard OAuth 2.0 and OpenID Connect protocols ensure compatibility 3. Vendor Assessment:

	* Formal evaluation process for all external dependencies * Regular review of vendor status, maintenance, and security * SLA monitoring and incident history tracking * Contractual protections for critical services 4. Architectural Principles: * Defense in depth with fallback capabilities for non-critical features * Core business logic remains within our controlled environment * External services primarily enhance rather than enable core functionality * Clear isolation boundaries around external dependencies 5. Risk Mitigation Strategies: * Regular dependency updates to maintain security and reliability * Comprehensive monitoring of all external service health * Circuit breakers implemented for graceful degradation * Cached responses where appropriate to handle temporary outages * Regular testing of fallback mechanisms Our approach balances the benefits of reliable external services with the need to maintain control over core functionality. By carefully selecting enterprise-grade providers, implementing appropriate architectural patterns, and maintaining fallback options, we minimize the risks associated with external dependencies while leveraging their capabilities to deliver a robust platform. .
1.37. What external services are you using in your widgets?	Our widgets are designed with minimal external dependencies to ensure reliability, performance, and data privacy: Core Widget Dependencies: 1. Content Delivery Networks (CDNs): * Amazon CloudFront for static asset delivery * Provides global edge caching for improved widget loading times * High-reliability service with 99.9% availability SLA * Fallback mechanisms for direct asset loading if CDN is unavailable 2. Font Services: * Google Fonts API for consistent typography (with local fallbacks) * Used only for standard fonts with high cache efficiency * Local font fallbacks ensure rendering if external service is unavailable * Configurable to use customer's self-hosted fonts instead

	3. Analytics Integration: * Minimal, privacy-focused analytics for widget performance monitoring * First-party analytics with no third-party tracking scripts * Performance metrics collection with anonymized data * Optional integration with customer's own analytics services (Google Analytics, Adobe Analytics) Widget Architecture Principles: 1. Self-Contained Design: * Core widget functionality operates without external runtime dependencies * All critical JavaScript and CSS bundled within the widget * No third-party JavaScript libraries loaded at runtime * Survey rendering and data collection operate independently of external services 2. Privacy-First Approach: * No third-party cookies or tracking mechanisms * No social media integrations or sharing buttons * No advertising or marketing technology integrations * GDPR-compliant data handling within the widget 3. Customer Control Options: * Self-hosting option for all widget assets * Configurable CSP (Content Security Policy) settings * Ability to disable external font loading * Custom domain configuration for widget resources 4. Performance Optimization: * Minimal network requests during widget initialization * Lightweight design with progressive enhancement * Offline capability for basic survey completion * Optimized asset loading and caching strategies Our widget architecture prioritizes independence from external services, ensuring that survey functionality remains reliable and performant even in restricted network environments. The limited external dependencies that do exist are carefully selected enterprise-grade services with fallback mechanisms to handle potential service disruptions.
Data privacy:	
1.38. Which kind of personal data of our end-users would you store on the platform?	Stored Data: Includes name, email, IP address, device and browser data, response metadata

Technical Due Diligence

DFL DIGITAL SPORTS

1.39. Which kind of personal data of our end-users would store in the devices (browser / mobile device)?	Data in Device: Cookies, page tags, tracking data.
1.40. In which country/countries do you store personal data of end-users of your European customers?	Storage Location: EU servers (Frankfurt, Ireland); some subprocessors in US for comms tools (Slack, Google Workspace)
1.41. Which other data center locations do you offer?	No
1.42. From which other countries can you (e.g. your support team) access the personal data of our end-users?	EU only
1.43. Please provide us with a complete list of your sub-processors (including affiliates). Such list shall also include the following information per sub-processor: location, location of data processing, provided services, list of accessed/processed data, purposes of data access/processing.	Subprocessors pursuant to Data Processing Agreement The Processor currently works with the following subcontractors and the Controller hereby agrees to their appointment. 1. Amazon Web Services Company: Amazon Web Services EMEA SARL (“AWS”), country of registration: Luxembourg Data processing activities: Data center Data location: Ireland and Frankfurt zone 2. ElasticSearch Company: Elasticsearch B.V., country of registration: The Netherlands Data processing activities: Search engine, indexing, search criteria. Data location: Ireland 3. Google Cloud Platform Company: Google Cloud EMEA Limited, country of registration: Ireland Data processing activities: Data center Data location: Frankfurt, Germany 4. Microsoft Azure: Cloud Computing Services Company: Microsoft Corporation, country of registration: France Data processing activities: Data center Data location: France

Technical Due Diligence

DFL DIGITAL SPORTS

1.44. How do you ensure the fulfilment of the rights of the data subjects, in particular the right to information pursuant to Art. 15 GDPR? And is it possible to export individual data records?	Rights Fulfillment: Export/deletion on request; manual deletion within 1 month of account closure
1.45. Do you provide an end-user audit of all user interactions? Can each end-user view and modify his/her usage history (e.g. voting history)?	Our privacy policy confirms that data subjects have the right to request access, correction, deletion, restriction, or blocking of their data at any time, in accordance with GDPR. Survey respondents (Survey Recipients) are requested to turn to the Survey Sender (the company) to request deletion. We act as a processor on behalf of the Survey Sender and process the data according to their instructions. We accept deletion requests via all e-mail channels posted on our website: support@zenloop.com, dpa@zenloop.com & privacy@zenloop.com
1.46. How does a user onboarding look like?	Joint Online Sessions
1.47. Can we set deletion deadlines manually? And do you offer automatic deletion routines and, if so, can we intervene manually (e.g. in the event of a legal dispute with the end-user that justifies further storage)?	Data deletion process communicated to zenloop and taken by zenloop. Automatic data anaonymisation post a specific amount of days.
1.48. Is it ensured that archived end-user data is not reactivated and merged with the existing end-user history upon an end user's return/new registration without his/her consent?	Yes, our platform is designed with strict data isolation mechanisms that prevent archived end-user data from being automatically reactivated or merged with new data without explicit consent: Data Isolation and Archiving Architecture: 1. Complete Data Separation: * Archived data is stored in separate database partitions from active data * Different access controls govern archived versus active data * Archive and production environments maintain strict separation * Unique identifiers are not automatically linked between archived and new records 2. Identity Management Safeguards: * New registrations generate new unique identifiers unconnected to archived profiles * Email addresses or other contact information do not automatically trigger data merging

	* Historical user records remain in archived state even if identifiable information matches * Correlation between archived and active records requires explicit administrative action 3. Consent-Based Reactivation Process: * Any potential data reactivation requires explicit, documented end-user consent * Consent collection follows GDPR standards for informed, specific, and unambiguous approval * Multi-step verification prevents accidental or unauthorized reactivation * Audit trails record all consent actions and administrative data operations 4. Technical Implementation: * Database schema design enforces separation of archived and active data * Dedicated APIs for archive management with strict access controls * Application logic prevents automatic historical data surfacing * Regular data isolation testing verifies effectiveness of separation controls 5. Data Processing Guidelines: * Clear documentation for customers on proper handling of archived data * Administrative interfaces clearly distinguish between archived and active records * Training materials emphasize consent requirements for data reactivation * Regular compliance reviews ensure adherence to data isolation policies This approach ensures that returning users or new registrations with potentially matching identifiers start with a clean slate and are not automatically associated with previously archived data. Any reconnection of historical data would only occur with clear, documented consent in compliance with applicable data protection regulations. .
1.49. Do you encrypt, mask and/or anonymize the personal data of end-users?	Yes, we implement multiple layers of data protection through encryption, masking, and anonymization to safeguard end-user personal data: Data Encryption: 1. Data in Transit:

	* All network communications secured via TLS 1.2+ encryption * API connections require HTTPS with strong cipher suites * Strict Transport Security (HSTS) enforced for all connections * Backend service-to-service communication encrypted even within private networks 2. Data at Rest: * Database-level encryption using AWS RDS encryption * Storage-level encryption for all S3 buckets containing user data * Encryption keys managed through AWS KMS * Backup files encrypted before storage * Application-level encryption for specific sensitive fields Data Masking: 1. User Interface Masking: * Partial masking of sensitive data in user interfaces (e.g., email addresses shown as j******@example.com) * Contextual masking based on user permissions and access needs * Configurable masking rules for different data types 2. Export and Reporting Masking: * Personal identifiers masked in exports unless specifically authorized * Configurable masking levels for different export purposes * Audit logs showing access to unmasked data * Preview mode showing masked data before export completion Data Anonymization: 1. Analytics Processing: * Personal identifiers removed from analytics datasets * Aggregation of data to prevent individual identification * Use of pseudonyms instead of direct identifiers where individual-level data is needed * Statistical controls to prevent re-identification through correlation 2. Benchmark and Cross-Customer Analytics: * Complete anonymization of data used for benchmarking * Removal of all customer-specific and personal identifiers * Minimum threshold requirements for data inclusion to prevent isolation * Regular reviews of anonymization effectiveness Implementation and Management:

	1. Data Classification Framework: * Systematic classification of data sensitivity levels * Protection mechanisms tied to data classification * Regular data inventory and classification audits * Automated scanning to identify unprotected sensitive data 2. Access Controls for Protected Data: * Role-based access to encryption keys and unmasked data * Just-in-time access for administrative functions * Comprehensive logging of all access to sensitive data * Regular access reviews and privilege right-sizing 3. Data Protection by Design: * Protection mechanisms built into data models and schemas * Default masking of sensitive fields in development environments * Automated testing to verify encryption and masking implementation * Regular security reviews of data protection mechanisms These multi-layered data protection measures ensure that personal data is appropriately secured throughout its lifecycle in our system, from collection through processing, storage, and eventual deletion or anonymization. .
1.50. Role and rights management: Describe the possibility to define dedicated functional and process-related roles and rights within the platform (access only to certain data and functions)	Our platform implements a sophisticated, multi-dimensional role and rights management system that enables precise control over data access and functional capabilities: Role-Based Access Control Framework: 1. Hierarchical Role Structure: * Pre-defined organizational roles (Admin, Manager, Analyst, Viewer) * Custom role creation with granular permission configuration * Role inheritance for efficient permission management * Role templates for common access patterns * Department-specific role customization 2. Fine-Grained Permission Model: * Modular permissions organized by functional area * Operation-specific rights (create, read, update, delete, execute) * Object-level permissions (surveys, responses, reports, integrations) * Feature-based access controls for premium capabilities * API access rights management Data Access Controls:

	1. Multi-Level Data Segmentation: * Organization-level data isolation for multi-tenant security * Department/team data segmentation within organizations * Project-based access controls for collaborative work * Survey-specific access permissions * Response data access filtered by attributes (e.g., region, product line) 2. Contextual Access Policies: * Time-based access restrictions for sensitive operations * Workflow state-dependent permissions * Approval workflows for sensitive operations * Temporary access grants with automatic expiration * IP-based access restrictions (optional) Implementation and Management: 1. Centralized Permission System: * Unified administration interface for permission management * Visual permission mapping to understand effective access * Batch permission updates for organizational changes * Permission audit logs tracking all changes * Conflict detection for contradictory permission assignments 2. User Management Integration: * Single Sign-On (SSO) integration with attribute-based role mapping * Active Directory/LDAP synchronization options * Automatic role assignment based on identity provider groups * Just-in-time role activation for privileged operations * Regular access recertification workflows 3. Self-Service Capabilities: * Delegated administration for departmental access management * Request and approval workflows for elevated access * User profile-based permission visibility * Access request justification and documentation Practical Implementation Examples: 1. Department-Specific Data Access: * Marketing team members can only view marketing-related surveys * Support team can access customer feedback but not internal employee surveys * Regional managers can only see data from their geographical territory * Product teams can filter responses by product line

	2. Functional Role Separation: * Survey creators can design but not publish surveys * Analysts can view data but not modify survey structures * Report creators can build dashboards without seeing individual responses * Integrators can manage API connections without accessing survey content This comprehensive access control system enables organizations to implement the principle of least privilege while providing the flexibility to align system access with business processes and organizational structures.
1.51. Do you use AI services as part of your platform?	Yes, we strategically incorporate AI services into our platform across multiple workflows, enhancing both analytical capabilities and content creation while maintaining appropriate human oversight: AI-Powered Capabilities: 1. Survey Creation and Design: * AI-assisted survey generation from simple prompts or keywords * Intelligent question formulation based on research objectives * Automated survey structure recommendations * Question optimization for response quality * Smart templates that adapt based on survey purpose 2. Content Design: * AI-powered email embed design generation * Visual layout optimization for different devices * Design suggestions based on brand guidelines * Personalized content creation for different audience segments * Automated A/B test variant generation 3. Feedback Analysis: * Natural Language Processing for sentiment analysis across multiple languages * Automated topic detection and categorization from free-text responses * Entity recognition to identify products, features, or locations mentioned * Intent classification to understand customer needs and pain points * Emotion detection for nuanced understanding beyond basic sentiment 4. Insight Generation:

	* Pattern recognition across large volumes of feedback * Trend identification and anomaly detection * Automated summarization of key themes from extensive feedback * Priority recommendation based on feedback impact analysis * Correlation detection between different feedback sources 5. Response Processing: * Language detection for multi-lingual survey support * Translation capabilities for global deployments * Content moderation for inappropriate or sensitive content * Response quality assessment Technical Implementation: 1. AI Services Architecture: * Python-based microservices dedicated to AI processing * Azure OpenAI integration for advanced language understanding * Custom-trained models for industry-specific terminology * Hybrid approach combining rule-based and ML techniques * Asynchronous processing to maintain platform performance 2. Responsible AI Framework: * Confidence scoring for all AI-generated insights and content * Clear indication of AI-generated content vs. human creation * Explicit requirement for human validation of AI results * Fallback mechanisms when confidence thresholds aren't met * Regular auditing of AI model performance and bias 3. Data Privacy Considerations: * AI processing occurs within our secure environment * No customer data is used to train models without explicit consent * Anonymization applied before AI processing where appropriate * Compliance with data sovereignty requirements for AI processing Customer Control and Transparency: 1. Configuration Options: * AI features can be enabled/disabled at organization level * Customization of confidence thresholds for different use cases * Ability to train on customer-specific terminology (optional) * Control over which workflows utilize AI assistance 2. Transparency Practices: * Clear labeling of AI-generated content in the interface * Explanation of methodology behind AI suggestions

	* Human review and editing capabilities for all AI-generated content * Documentation of AI capabilities and limitations Our approach to AI integration emphasizes augmenting human creativity and analysis rather than replacing it. While AI services provide valuable assistance in content creation and data analysis, we maintain a "human-in-the-loop" philosophy where customers review, edit, and approve AI-generated content to ensure quality, brand alignment, and contextual appropriateness.
1.52. Would you use personal data of our end users for own purposes (e.g. optimization of the platform, training of an AI service)? If so, do you provide your customers with an opt out opportunity?	No, we do not use personal data of your end users for our own purposes such as platform optimization or AI service training. We maintain strict data isolation principles and respect the data controller relationship: Data Processing Principles: 1. Strict Purpose Limitation: * All customer data, including end-user personal data, is used solely for providing the contracted services * We process data only on behalf of our customers as per their instructions * No repurposing of personal data for our internal product development or optimization * No use of customer data for training our AI models without explicit consent * All data processing activities are documented in our Data Processing Agreement 2. Data Isolation and Governance: * Strong logical separation between customer data environments * Technical measures preventing cross-customer data access * Strict internal access controls limiting employee access to customer data * Regular access reviews and comprehensive audit logging * Formal data governance processes overseeing all data handling Analytics and Platform Improvement: 1. Anonymized Usage Data: * Platform analytics rely solely on anonymized, aggregate usage metrics * No personal data is included in our internal analytics systems * Strict anonymization processes remove all identifying information * Statistical controls prevent re-identification through analysis * Aggregate metrics focus on system performance, not individual behavior

	2. AI Model Training: * Our AI models are pre-trained on public or licensed datasets, not customer data * Fine-tuning uses synthetic data or explicitly consented datasets only * No extraction of customer text responses for model improvement * Model evaluation uses anonymized performance metrics only Customer Control Framework: 1. Opt-Out Implementation: * While we don't use personal data, we still provide opt-out mechanisms as a best practice * System-wide configuration to exclude specific surveys from even anonymized analysis * Organization-level settings for aggregate analytics participation * Data Processing Agreement specifies all permitted data uses * Implementation of data minimization principles throughout the platform 2. Transparency and Documentation: * Clear documentation of all data processing activities * Regular data processing reports available to customers * Data flow diagrams depicting all data handling processes * Independent audits validating our data processing claims We operate under a privacy-first philosophy where we view ourselves as stewards of your data, not owners. Your end-users' personal data remains under your control, and we process it solely for providing the services you've contracted us to deliver.
1.53. Insofar as anonymized data is used for such purposes, how does anonymization work and is anonymization carried out exclusively for the use of this data for such purposes?	While we don't use personally identifiable information for our own purposes, we do implement robust anonymization for certain limited uses such as benchmarking and platform analytics. Our anonymization processes are comprehensive and purpose-specific: Anonymization Methodology: 1. Technical Anonymization Process: * Removal of all direct identifiers (names, email addresses, user IDs, IP addresses) * Replacement of unique identifiers with non-reversible hash values * Aggregation of data to prevent individual response identification

	* Truncation of free-text responses to remove potentially identifying content * Generalization of demographic or geographic data to broad categories * Noise addition techniques for certain numeric values to prevent fingerprinting * Minimum threshold requirements preventing isolation of unique responses 2. Differential Privacy Implementation: * Statistical techniques ensuring individual contributions cannot be isolated * Formal privacy budgeting for aggregate analytics * Mathematical guarantees against re-identification through inference * Randomization techniques protecting outlier values * Automated verification of anonymization effectiveness Purpose-Specific Anonymization: 1. Benchmarking Anonymization: * Industry benchmarks created only from anonymized, aggregated data * Multiple organizational datasets combined to prevent company identification * Minimum participation thresholds for inclusion in benchmark datasets * Temporal shifts to prevent time-based re-identification * Sector-level rather than company-level comparisons 2. Platform Analytics Anonymization: * System performance analytics use anonymized interaction patterns * Feature usage tracking at aggregate level only * Error tracking with personal data stripped from logs * Workflow efficiency analysis at statistical level * Strict separation of analytical and operational data stores Exclusivity of Anonymization Purpose: 1. Purpose Limitation: * Anonymization processes are specifically designed for and only applied to data used for benchmarking and analytics * Different anonymization techniques applied based on specific use case * Anonymization is not used as a general-purpose tool to enable broader data use * Each anonymization purpose has its own documented methodology and controls

	2. Governance Controls: * Formal approval process for any new anonymized data use case * Regular auditing of anonymization effectiveness * Periodic review of purpose limitations * Automated monitoring for potential anonymization failures * Clear documentation of all anonymized data flows 3. Customer Transparency: * Detailed explanation of anonymization methodologies available to customers * Clear disclosure of all anonymized data use cases in service documentation * Organization-level opt-out available for benchmark inclusion * Regular reporting on anonymization practices and improvements Our approach ensures that anonymization is not treated as a blanket permission for unrestricted data use, but rather as a carefully controlled process applied only for specific, limited purposes that provide value back to our customer community through improved benchmarking and platform performance.
1.54. Describe the multi-tenancy of the platform (with regard to your other customers as well as regarding the following use case: digital products of DFL group are operated by different DFL entities and data sets may not be merged without the consent of the end user).	Our platform implements a robust multi-tenancy architecture: Multi-Tenant Architecture: 1. Hierarchical Tenant Model: * Root-level separation between different customer organizations * Sub-tenant capability for different entities within a parent organization * Each tenant maintains its own isolated data environment * Granular access controls at both tenant and sub-tenant levels * Configurable relationships between tenants based on business requirements 2. Technical Implementation: * Database-level segregation with tenant-specific partitioning strategies * PostgreSQL row-level security policies enforcing tenant isolation * Organization ID required for all database queries with automatic filtering * Application-level tenant context validation on every request * API authentication and authorization tied to specific tenant scopes Data Segregation Guarantees:

	1. Cross-Customer Isolation: * Complete separation between different customer organizations * No data sharing or visibility across customer boundaries * Independent encryption keys for each customer's data * Separate backup and recovery processes * Customer-specific configuration and customization 2. Sub-Entity Separation (DFL Use Case): * Different DFL entities can operate as separate sub-tenants * Data collected by one DFL entity remains isolated from other DFL entities * No automatic data merging or cross-entity visibility * Entity-specific user access rights with no default cross-entity access * Custom permission sets defining exactly what data each entity can access Consent-Based Data Sharing: 1. Explicit Consent Framework: * Data sharing between entities requires explicit end-user consent * Granular consent options for specific data types and purposes * Temporal limitations on consent validity * Comprehensive consent tracking and audit trail * Ability to revoke previously granted consent 2. Shared Data Implementation: * When consent is granted, data sharing occurs through controlled APIs * Shared data is clearly marked with source entity attribution * Access logs record all cross-entity data accesses * Selective field sharing based on consent specifics * Temporal controls enforcing consent expiration Multi-Tenancy Administration: 1. Tenant Management: * Centralized tenant configuration for parent organizations * Delegated administration for sub-tenants * Tenant relationship visualization and management * Tenant-specific feature configuration * Audit logs for all tenant management actions 2. Tenant Data Controls: * Tenant-specific data retention policies * Independent backup and restore capabilities

	* Data export controls at tenant level * Tenant-specific compliance settings * Custom notification and approval workflows for data operations This multi-tenancy architecture specifically addresses the DFL use case by ensuring that data collected by different DFL entities remains strictly separated unless explicit consent is provided for merging or sharing. Each entity operates in its own secure environment with full control over its data, while the platform provides the necessary consent management and selective sharing capabilities for appropriate cross-entity collaboration when authorized.
General:	
1.55. How easy is it to set up and configure the LiveLike platform?	Our zenloop platform is designed for straightforward setup and configuration, with a focus on self-service capabilities balanced with enterprise-grade customization options: Initial Setup Process: 1. Account Provisioning: * Simple signup process with immediate access to the platform * Guided organization setup with branding and preference configuration * Role-based user invitation system for team onboarding * Quick-start templates for common feedback collection scenarios * Interactive tutorial walks through core platform features 2. Survey Implementation: * Intuitive survey builder with drag-and-drop functionality * Pre-built survey templates for various use cases (NPS, CSAT, CES) * Customizable survey designs to match brand guidelines * Mobile-responsive preview showing how surveys appear across devices * Simple embedding options via JavaScript, direct link, or email 3. Integration Options: * JavaScript snippet for website integration (copy-paste implementation) * RESTful API for custom implementations and complex use cases * Pre-built connectors for popular CRMs, email providers, and CX platforms * Webhook support for event-driven architecture * Detailed API documentation with implementation examples Configuration Flexibility:

Technical Due Diligence

DFL DIGITAL SPORTS

	1. Customer Journey Mapping: * Visual journey mapping tool to identify feedback touchpoints * Configurable trigger rules based on user behavior * Event-based survey display options * Multi-channel feedback collection (web, email, in-app) * Customizable survey timing and frequency settings 2. Response Handling: * Flexible alert and notification configuration * Customizable workflows for different response types * Team routing rules based on feedback content * Automated response templates for acknowledgment * SLA management for response handling 3. Analysis Configuration: * Customizable dashboards for different stakeholder needs * Configurable metrics and KPIs * Custom tag taxonomies for response categorization * Flexible filtering and segmentation options * Scheduled report configuration Implementation Support: 1. Self-Service Resources: * Comprehensive knowledge base with setup guides * Video tutorials covering all major features * Interactive sample configurations for common scenarios * Community forum for peer support and best practices * Regular webinars on optimization techniques 2. Enterprise Implementation: * Dedicated implementation specialists for enterprise customers * Joint planning sessions to define implementation roadmap * Custom integration support for complex environments * Data migration assistance from legacy systems * Phased rollout planning for large organizations Our platform balances ease of use with enterprise flexibility, allowing organizations of all sizes to get started quickly while providing the depth of configuration needed for sophisticated feedback programs. Most standard implementations can be completed in days rather than weeks, with enterprise integrations typically completed within 2-4 weeks depending on complexity.

1.56. How fast and efficient is data processing on the platform?	Our platform delivers exceptional data processing performance through a carefully optimized architecture designed for speed, scalability, and efficiency: Real-Time Data Processing: 1. Survey Response Processing: * Near-instantaneous ingestion of survey responses (typically <100ms) * Real-time data validation and sanitization * Immediate storage in high-performance database systems * Concurrent processing of thousands of simultaneous responses * Asynchronous processing pipeline for analysis without response delay 2. Analysis Pipeline Performance: * Sentiment analysis completed within seconds of response receipt * Topic categorization and tagging processed in near real-time * Automated classification typically completed in under 5 seconds * Dashboard metrics updated in real-time as responses arrive * Alert triggers evaluated instantaneously for time-sensitive notifications Technology Optimizations: 1. Elixir Backend Advantages: * Erlang VM (BEAM) provides exceptional concurrency for handling large volumes * Actor model enables efficient parallel processing of response data * Low-latency message passing between system components * Fault tolerance with automatic recovery for continuous processing * Efficient memory utilization enabling performance at scale 2. Database Performance: * PostgreSQL with optimized partitioning strategy for efficient queries * Sophisticated indexing based on common access patterns * Connection pooling for optimal resource utilization * Query optimization for high-volume operations * ElasticSearch for fast full-text search and complex querying 3. Processing Architecture: * Event-driven design for optimized workload distribution * Multi-stage processing pipeline with targeted optimizations * Stateless application design enabling horizontal scaling * Efficient caching strategies for frequently accessed data * Background job prioritization for optimal resource allocation

	Performance Metrics: 1. Throughput Capabilities: * Capable of processing hundreds of thousands of responses per hour * Load-balanced to handle traffic spikes during high-volume campaigns * Linear scaling with additional resources for predictable performance * Processing bandwidth consistently exceeds typical customer requirements * Capacity planning ensures headroom for peak periods 2. Analysis Performance: * Standard analytical queries return results in milliseconds * Complex aggregate reports typically generated in under 3 seconds * Large dataset exports (100,000+ records) processed in under 30 seconds * AI-enhanced analysis completed in under 10 seconds for most responses * Historical trend analysis optimized through materialized views 3. End-User Experience: * Dashboard loading times under 1 second for standard views * Filter applications reflected instantly in user interface * Report generation typically completed in 2-5 seconds * Dynamic recalculation of metrics as new data arrives * Smooth pagination and scrolling through large result sets Our platform's performance is a result of intentional architecture decisions, particularly the selection of Elixir for its concurrency model, combined with optimized database design and efficient processing pipelines. This ensures that customers experience responsive performance even with large volumes of feedback data.
1.57. How scalable is the platform to handle future growth and increased payload?	Our platform is architected from the ground up for exceptional scalability, with proven capabilities to handle significant growth in both user base and processing volume: Architectural Scalability: 1. Cloud-Native Infrastructure: * AWS and GCP infrastructure leveraging elastic scaling capabilities * Multi-Availability Zone deployment for reliability and load distribution * Containerized application components enabling rapid scaling * Infrastructure-as-Code ensuring consistent environment expansion * Auto-scaling groups that adjust capacity based on demand patterns

	2. Modular Design: * Microservices-inspired architecture with clear component boundaries * Independent scaling of individual services based on specific load patterns * Loose coupling between components via message-based communication * Stateless application design supporting horizontal scaling * Service discovery for dynamic resource allocation 3. Processing Pipeline Scalability: * Event-driven architecture enabling asynchronous processing * Parallel processing of independent operations * Elixir/OTP supervision trees for resilient concurrent processing * Background job queuing with prioritization and rate limiting * Resource pooling for optimal utilization during peak loads Data Layer Scalability: 1. Database Scalability: * PostgreSQL with sophisticated partitioning for horizontal scaling * Time-based partitioning for efficient historical data management * Organization-based partitioning for multi-tenant isolation * Read replicas for scaling query-intensive workloads * Connection pooling to optimize database resource utilization 2. Search and Analytics Scaling: * ElasticSearch cluster supporting sharding and replication * Distributed search capability across multiple nodes * Efficient indexing strategies for large-scale datasets * Optimized query patterns for high-volume operations * Aggregation pipeline designed for large result sets Proven Scaling Capabilities: 1. Current Production Scale: * Successfully handling millions of survey responses daily * Supporting thousands of concurrent users across customer organizations * Processing terabytes of feedback data with consistent performance * Managing tens of thousands of active surveys simultaneously * Delivering sub-second response times for common operations 2. Growth Headroom: * Architecture tested to 10x current peak loads through load testing * Linear scaling characteristics validated through performance modeling

	* Cost-efficient scaling through right-sized resource allocation * No architectural bottlenecks identified in stress testing * Ongoing performance optimization reducing resource requirements Scaling Methodology: 1. Proactive Capacity Management: * Continuous monitoring of resource utilization trends * Predictive scaling based on historical patterns * Regular load testing with projected future volumes * Capacity planning aligned with customer growth forecasts * Performance efficiency reviews identifying optimization opportunities 2. Scaling Operations: * Zero-downtime scaling for all components * Automated scaling triggers based on defined thresholds * Graceful degradation strategies for extreme load scenarios * Regional scaling options for global performance optimization * Cost-optimized scaling balancing performance and efficiency Our platform's scalability has been proven in production environments with some of our largest customers, demonstrating the ability to grow seamlessly from startup scale to enterprise volumes without architecture changes. This scalability ensures that as your feedback program grows, our platform can expand proportionally to maintain consistent performance.
1.58. How reliable and stable is the platform, especially in terms of downtime and maintenance? What are your current SLAs? Do you have a status page?	Our platform maintains high reliability and stability through robust architecture, proactive monitoring, and careful maintenance practices: System Reliability: 1. Availability Commitment: * SLA guarantee of 99.5% uptime measured annually * Translates to maximum allowed downtime of approximately 43.8 hours per year * Actual historical performance exceeding 99.9% uptime (less than 9 hours downtime per year) * SLA calculation excludes scheduled maintenance windows * Performance against SLA is actively monitored and reported 2. Infrastructure Reliability: * Redundant components eliminating single points of failure

	* Automatic failover for critical services * Load balancing across multiple application instances Maintenance Approach: 1. Scheduled Maintenance: * Regular maintenance windows scheduled outside business hours: - Typically between 18:00 and 22:00 CET - Maximum scheduled downtime of 1.5 hours per week - Additional maintenance windows on Tuesdays and Thursdays (10:00-10:20 CET) * Advanced notification for all planned maintenance * Rolling updates to minimize or eliminate service interruption * Transparent communication of maintenance activities and expected impact 2. Release Management: * Zero-downtime deployment for most platform updates * Automated rollback capabilities for problematic releases * Thorough pre-release testing to minimize production issues * Staged deployment process across environments Incident Management: 1. Issue Detection and Response: * 24/7 automated monitoring and alerting * Dedicated operations team during business hours * On-call engineering support outside business hours * Escalation procedures for critical issues * Post-incident analysis for continuous improvement 2. Communication Protocol: * Proactive notification of service disruptions * Regular status updates during ongoing incidents * Transparent post-incident reporting * SLA-based response time commitments * Multiple communication channels for status updates Platform Status Visibility: 1. Current Status Communication: * Apar from public-facing status page (https://status.zenloop.com), we provide: - Email notifications for planned maintenance

	- Direct alerts for service disruptions affecting specific customers - Regular status reports for enterprise customers - Incident notifications via email and in-platform alerts * We're actively developing a public status page for launch in Q3 2025 2. Status Monitoring: * Comprehensive internal monitoring dashboard * Real-time performance metrics tracking * Historical reliability reporting * Component-level status visibility * Trend analysis for proactive issue prevention Our platform's reliability is built on a foundation of redundant architecture, careful maintenance practices, and proactive monitoring. We continuously refine our approach to minimize disruptions while balancing the need for regular updates and improvements. While we do not currently offer a public status page, we provide transparent and timely communication about any service impacts through direct customer channels.
1.59. How secure is the platform in terms of storing and processing sensitive data?	Our platform implements comprehensive security measures across all layers to ensure the protection of sensitive data throughout its lifecycle: Data Protection Architecture: 1. Infrastructure Security: * EU-only data storage in compliant data centers (AWS Frankfurt/Ireland, GCP Frankfurt) * Virtual private cloud (VPC) implementation with strict network segmentation * Multiple security groups and network ACLs controlling traffic flow * DDoS protection at network and application layers * Vulnerability scanning and penetration testing 2. Data Encryption: * TLS 1.2+ with strong cipher suites for all data in transit * HTTPS enforced for all communications with strict HSTS * Database encryption at rest using AWS RDS encryption * S3 bucket encryption for stored files and backups 3. Access Control: * Role-Based Access Control (RBAC) with principle of least privilege * Multi-factor authentication for administrative access * Fine-grained permissions model for internal access * Just-in-time access for operations requiring elevated privileges

	* Comprehensive audit logging of all access to sensitive data * Regular access reviews and privilege right-sizing Compliance and Governance: 1. GDPR Compliance: * Full compliance with Article 32 GDPR requirements * Documented Technical and Organizational Measures (TOMs) * Data protection by design and default principles * Data minimization practices throughout the platform * Privacy impact assessments for new features * Formalized incident response process for data breaches 2. Secure Development: * Secure coding practices integrated into development lifecycle * Regular security training for all development staff * Automated security scanning in CI/CD pipeline * Dependency vulnerability monitoring * Code reviews with security focus * Threat modeling for new features 3. Security Monitoring: * Real-time security event monitoring and alerting * Automated anomaly detection for unusual access patterns * Log aggregation and analysis for security events * Regular security posture assessments * AWS security services (GuardDuty, Config, CloudTrail) * Intrusion detection systems monitoring for suspicious activity Data Handling Practices: 1. Data Segregation: * Strong multi-tenant isolation preventing cross-customer data access * Database partitioning by organization for additional separation * Logical separation of data processing environments * Strict controls on data movement between environments * Separate processing pipelines for each customer's data 2. Sensitive Data Management: * Data classification framework identifying sensitivity levels * Automated detection of potentially sensitive data * Special handling procedures for high-sensitivity information * Data anonymization for benchmarking and analytics * Secure data deletion processes when required * Minimal retention periods aligned with business needs

	3. Third-Party Risk Management: * Vendor security assessment process * Limited access for third-party services * Contractual security requirements for all vendors * Regular review of vendor security practices * Data processing agreements with all subprocessors Our security approach aligns with industry best practices and regulatory requirements, with a particular focus on the protection of sensitive feedback data. We maintain a defense-in-depth strategy that implements multiple layers of controls, ensuring that the security of your data is protected against a wide range of potential threats.
1.60. How well does the platform integrate with other systems and applications the company already uses?	Our platform offers extensive integration capabilities designed to seamlessly connect with your existing technology ecosystem through multiple flexible approaches: Integration Architecture: 1. API-First Design: * Comprehensive RESTful API covering all platform functionality * Well-documented OpenAPI/Swagger specifications * Consistent API patterns for predictable implementation * Versioned APIs ensuring backward compatibility * Authentication via OAuth 2.0 or API keys * Rate limiting with enterprise-appropriate thresholds 2. Event-Driven Integration: * Webhook system for real-time event notifications * Customizable event triggers for different business processes * Configurable payload formats to match recipient systems * Reliable delivery with retry mechanisms * Event filtering for targeted integrations * Testing tools for webhook endpoint validation 3. Data Exchange Formats: * JSON as primary data format for modern integrations * CSV exports for data analysis tools * Customizable data mapping for field alignment * Transformation capabilities for format conversion * Batch and incremental data synchronization options Pre-Built Integrations:

	1. CRM Systems: * Salesforce - bi-directional integration with custom object mapping * HubSpot - contact enrichment and feedback synchronization * Microsoft Dynamics - survey response integration * Zendesk - ticket creation from feedback and survey linking * Custom CRM connectors available for other platforms 2. Marketing and Communication: * Email service providers (Inxmail, Emarsys, Mailchimp) 3. Enterprise Systems: * Support desk integration (Zendesk, Freshdesk, ServiceNow) Integration Implementation: 1. Developer Experience: * Comprehensive API documentation with examples * Interactive API explorer for testing * Client libraries for popular programming languages * Implementation guides for common integration patterns * Developer support for custom integration challenges 2. No-Code/Low-Code Options: * Integration platform support (Zapier, Integromat) * Configurable integration templates * Visual integration builders for common scenarios * Drag-and-drop webhook configuration * Pre-built automation recipes 3. Enterprise Integration Support: * Custom integration development for complex scenarios * Technical consultation for integration architecture * Implementation support from our integration specialists * Performance optimization for high-volume integrations * Integration monitoring and troubleshooting Integration Security: 1. Secure Connection Methods: * TLS encryption for all API communications * IP whitelisting options for API access * Signed webhook payloads ensuring authenticity * Granular API permissions model

	* Secure credential management 2. Data Protection in Integrations: * Field-level control over data shared in integrations * Audit logging of all integration activities * Data minimization in integrated workflows * Compliance with data protection requirements across systems Our flexible integration approach ensures that zenloop can connect with virtually any existing system in your technology stack, enabling seamless data flow and unified customer experience management across platforms.
1.61. How comprehensive are the platform's personalization options to meet individual requirements?	Our platform offers extensive personalization capabilities at multiple levels, enabling you to create tailored experiences that align with your brand and meet specific business requirements: Survey Design Personalization: 1. Visual Customization: * Comprehensive branding controls (colors, fonts, logos, imagery) * Custom CSS for pixel-perfect design alignment * Responsive layouts adapting to different devices and screen sizes * Theme templates for consistent brand application * White-labeling options removing all zenloop branding * Custom background images and visual elements 2. Content Personalization: * Dynamic text replacement using customer data variables * Personalized greetings and references in survey content * Conditional question display based on customer attributes * Language adaptation based on user preferences * Region-specific content variations * Custom thank you messages based on response type 3. Survey Flow Customization: * Advanced branching logic creating personalized paths * Skip logic hiding irrelevant questions * Question piping to reference previous answers * Custom scoring models for different survey types * Personalized follow-up questions based on sentiment * Exit points with different destinations based on responses Deployment Personalization:

	1. Targeting and Triggering: * Behavioral triggers based on user actions * Time-based triggers for optimal survey timing * Segment-based targeting for relevant audiences * Journey stage targeting for contextual feedback * Exclusion rules preventing survey fatigue * Multi-channel coordination for consistent experiences 2. Delivery Customization: * Channel-specific formatting (web, email, in-app) * Personalized invitation messages * Custom sender names and email addresses * Mobile-optimized layouts * Custom embedding options for website integration * API-driven custom deployment for unique scenarios 3. Timing and Frequency: * Customizable display timing and delays * Frequency capping at user and segment levels * Optimal time delivery based on historical engagement * Re-engagement rules for incomplete surveys * Reminder configuration and scheduling * Survey rotation for multiple feedback programs Data Collection Personalization: 1. Question Customization: * Custom question types beyond standard templates * Industry-specific question libraries * Personalized rating scales and labels * Custom validation rules for specific data formats * Pre-filled answers from known customer data * Context-adaptive question phrasing 2. Response Enhancement: * Custom data capture fields for specific business needs * Hidden fields passing contextual information * UTM parameter capture for marketing attribution * Device and environment data collection * Custom metadata tagging for advanced analysis * Integration with CRM data for enriched responses Analysis and Reporting Personalization: 1. Dashboard Customization:

	* Configurable dashboard layouts for different roles * Custom metrics and KPIs beyond standard NPS/CSAT * Personalized report distribution and scheduling * Team-specific views of relevant feedback * Custom data visualizations for specific insights * Threshold-based highlighting and alerting 2. Tagging and Categorization: * Custom tagging taxonomies for your business * Industry-specific categorization models * AI-assisted tagging with custom training * Hierarchical tag structures for detailed analysis * Multi-dimensional categorization frameworks * Personalized tag recommendations This comprehensive personalization framework ensures that you can create feedback experiences that feel native to your brand, relevant to your customers, and aligned with your specific business processes and goals. From visual design to functional behavior, data collection to analysis, our platform can be tailored to meet your unique requirements.
1.62. How well does the platform work on different devices and channels, including mobile devices?	Our platform is designed with a multi-channel, device-agnostic approach that ensures consistent functionality and optimal user experience across various touchpoints: Cross-Device Compatibility: 1. Responsive Design Architecture: * Fully responsive survey layouts that adapt to any screen size * Mobile-first design approach ensuring excellent small-screen experiences * Touch-optimized interactions for mobile and tablet users * Automatic scaling of visual elements for different device densities * Optimized form controls for touchscreen input * Accelerated Mobile Pages (AMP) support for lightning-fast mobile loading 2. Device-Specific Optimizations: * Tailored layouts for smartphones, tablets, and desktops * Portrait and landscape orientation support on mobile devices * Keyboard-friendly navigation for desktop users * Touch target sizing meeting accessibility guidelines * Bandwidth-aware resource loading for varying connection speeds

	* Battery-efficient operation on mobile devices 3. Browser Compatibility: * Support for all modern browsers (Chrome, Firefox, Safari, Edge) * Graceful degradation for older browser versions * Consistent rendering across browser engines * Feature detection with appropriate fallbacks * Regular compatibility testing across browser matrix * Progressive enhancement approach for newer devices Multi-Channel Implementation: 1. Web Channel Integration: * JavaScript widget for website embedding * Modal, inline, and side-panel display options * Exit-intent and time-based triggers * Page-specific targeting rules * Non-intrusive display patterns preserving user experience * Performance optimization minimizing impact on page load times 2. Email Channel Capabilities: * Embedded surveys directly within email content * HTML email templates optimized for major email clients * Click-through survey links with context preservation * Personalized email invitations with dynamic content * Mobile-optimized email layouts * Tracking and analytics for email-based surveys 3. In-App Experience: * While we don't offer a native SDK, our platform provides: * RESTful API for custom in-app implementation * WebView integration guidelines for mobile apps * JavaScript API for controlling survey behavior in apps * Custom events support for app-specific triggers * Deep linking capabilities for cross-channel journeys * Integration examples for popular app frameworks 4. Omnichannel Coordination: * Consistent branding across all channels * Cross-channel response tracking and consolidation * Channel-aware question formatting * Unified reporting across all collection points * Channel comparison analytics * Coordinated frequency management across touchpoints

	Technical Implementation: 1. Mobile Web Approach: * Lightweight implementation minimizing performance impact * Offline support with response caching when possible * Optimized assets for faster mobile loading * Touch event handling for natural mobile interactions * Viewport-aware rendering and positioning * Minimal network requests to conserve mobile data 2. Cross-Platform Consistency: * Unified codebase ensuring consistent behavior * Comprehensive testing across device matrix * Standardized data collection regardless of channel * Consistent validation and error handling * Accessibility compliance across all platforms * Synchronized styling with platform-appropriate adjustments Our multi-channel, device-agnostic approach ensures that your customers can provide feedback whenever and wherever it's most convenient for them, with a consistently high-quality experience regardless of device or channel. While we don't currently offer a native mobile SDK, our web-based approach and API capabilities provide excellent flexibility for mobile integration scenarios.
1.63. How fast and smooth is the platform's performance when processing large amounts of data?	The platform leverages Elixir for concurrent processing and ElasticSearch for fast search and filtering. Performance remains stable even with high data volumes. Dashboard filtering and response retrieval are near-instant in tested environments.
1.64. How many and what type of data sources are supported by the platform to provide comprehensive insights?	Our platform supports a diverse range of data sources that can be combined to create rich, contextual insights beyond standard survey responses: Primary Data Collection Sources: 1. Direct Feedback Channels: * Web-embedded surveys (modal, inline, and side-panel formats) * Email surveys (embedded and click-through options) * Link-based surveys for flexible distribution * QR code surveys for physical touchpoints * In-app feedback via API integration * Post-transaction surveys triggered by events 2. Feedback Types and Formats: * Structured feedback (NPS, CSAT, CES scores)

	* Free-text responses and comments * Rating scales (star, numeric, emoji) * Multiple-choice questions * Open-ended text questions * Media feedback (optional image uploads) Contextual Data Enrichment: 1. User Attribute Data: * Demographic information from CRM systems * Customer segmentation attributes * Loyalty program status and history * Account lifecycle stage indicators * Purchase and transaction history * Support and service interaction records 2. Behavioral Data: * Custom properties passed via URL parameters * User journey stage markers * Previous interaction history * Feature usage patterns (through API) * Engagement metrics and touchpoint history * Action timestamps and sequence information 3. Operational Context: * Product/service identifiers * Transaction IDs and order information * Support ticket references * Agent or team identifiers * Channel and touchpoint indicators * Regional and locational data Integration Data Sources: 1. CRM System Integration: * Customer profile enrichment from Salesforce, HubSpot, etc. * Account history and relationship data * Team assignments and ownership information * Custom object data relevant to feedback context * Opportunity and pipeline stage information * Contact engagement history 2. Business System Connections: * Product catalog information * Inventory and availability data

	* Pricing and promotion details * Order and fulfillment status * Service level agreement information * Account and subscription details 3. Marketing Platform Data: * Campaign attribution data * Email marketing interaction history * Marketing automation workflow states * Content engagement metrics * Acquisition source and channel information * Customer journey position indicators Analysis Enhancement Sources: 1. Historical Trend Data: * Previous survey responses from same users * Longitudinal analysis of changing sentiment * Historical benchmark comparisons * Seasonal pattern data * Response trend indicators * Time-based correlation analysis 2. Aggregation and Segmentation: * Industry benchmark comparisons * Peer group performance metrics * Regional and geographic segmentation * Demographic cohort analysis * Product/service category comparisons * Channel performance comparisons While our platform's primary data source is direct feedback through surveys, our flexible architecture allows for significant enrichment through these additional data sources, creating a comprehensive context for analysis and insight generation. This approach transforms isolated feedback points into rich, actionable insights by connecting them with relevant business and customer data.
1.65. What widgets can I use without a LiveLike login?	Our platform offers several widget options that can be used without requiring end users to log in: Public Access Survey Widgets: 1. Web Embed Surveys: * JavaScript-based embed widgets that can be added to any website

	* No authentication required for end users to provide feedback * Fully responsive design works across all devices * Customizable appearance to match your brand identity * Various display formats including modal popups, inline embeds, and side panels * Trigger options based on time, scroll depth, or user actions 2. Direct Link Surveys: * Standalone survey pages accessible via direct URL * Perfect for sharing via email, SMS, QR codes, or social media * No login required to access and complete * Custom URL options for branded links * Secure access even without authentication * Mobile-optimized for completion on any device 3. Email Embedded Surveys: * Surveys that appear directly within email content * Recipients can respond without leaving their email client * No login or redirect required * Initial question embedded with follow-up questions on landing page * Personalized content based on recipient data * Tracking parameters automatically included 4. Feedback Button Widgets: * Persistent feedback buttons that can be added to any webpage * Single-click access to feedback forms * No authentication required to submit feedback * Customizable appearance and positioning * Mobile-responsive design * Configurable to collect different types of feedback Data Collection Without Login: 1. Anonymous Feedback Collection: * All public-facing widgets support anonymous feedback collection * Optional identification fields if desired * Custom parameters can pass context without requiring login * Device and browser information collected automatically * IP-based geolocation (generalized for privacy) * Session tracking for multi-page experiences 2. Contextual Data Enrichment: * URL parameter support to add context to anonymous submissions * Custom hidden fields for additional metadata * UTM parameter capture for marketing attribution

	* Referrer tracking to understand feedback source * Timestamp and channel information automatically added * Custom identifiers can be passed without requiring login User Experience: 1. Seamless Access: * Zero-friction experience with no authentication barriers * Single-page design minimizing load times * Progress indicators for multi-question surveys * Save and resume functionality via browser storage * Optimized for both desktop and mobile completion * Accessibility compliance for all users 2. Response Management: * Automatic "thank you" messaging after submission * Redirect options to specific pages after completion * Follow-up content based on response type * Social sharing options after submission * Download or email receipt of responses if desired These authentication-free widgets are designed to maximize response rates by removing friction from the feedback process while still collecting rich, contextual data that drives valuable insights.
1.66. How is the frontend performance (Google Core Web vitals) for the widgets?	Our widgets are engineered with performance as a core design principle, optimized specifically for excellent Core Web Vitals metrics while minimizing impact on host pages: Core Web Vitals Performance: 1. Largest Contentful Paint (LCP): * Widgets are designed to load quickly with optimized assets * Typical LCP for modal surveys under 1.2 seconds on desktop * Mobile LCP typically under 2.4 seconds on 4G connections * Critical rendering path optimization reduces initial load time * Progressive loading prioritizes visible content first * Image optimization with next-gen formats and appropriate sizing 2. First Input Delay (FID) / Interaction to Next Paint (INP): * Minimal JavaScript execution on the main thread * Event handlers optimized for quick response * Debounced and throttled event listeners for efficiency * Typical FID under 100ms for interactive elements * Predictive pre-loading of subsequent survey steps

	* Efficient DOM manipulation minimizing layout shifts 3. Cumulative Layout Shift (CLS): * Widget containers pre-allocated with correct dimensions * Fixed size reservations for dynamic content * Proper aspect ratio management for multimedia elements * Typical CLS scores below 0.1 for embedded surveys * Careful management of font loading to prevent text shifts * Stable positioning for modal and slide-in widgets Performance Optimization Techniques: 1. Resource Efficiency: * Minimal bundle size (core widget under 80KB gzipped) * Code splitting to load only necessary components * Tree-shaking to eliminate unused code * Lazy loading for non-critical survey elements * Efficient asset caching strategy * CDN delivery for optimal geographic performance 2. Loading Strategy: * Asynchronous loading by default to prevent host page blocking * Configurable deferred loading until idle time * Progressive enhancement approach for core functionality * Server-side rendering support for critical content * Preconnect and resource hints for performance optimization * Optional preloading configurations for anticipated surveys 3. Rendering Optimization: * Efficient DOM operations minimizing reflows * Virtualized rendering for long surveys * GPU-accelerated animations for smooth transitions * Minimal CSS with optimized selectors * Efficient use of will-change for animation performance * Responsive transitions maintaining 60fps frame rate Host Page Considerations: 1. Implementation Guidance: * Best practice documentation for optimal integration * Performance impact analysis tools for implementation testing * Recommended placement guidelines for minimal disruption * Host page optimization suggestions for compatible configurations * Integration patterns optimized for different CMS platforms * Timing configuration to avoid critical rendering path conflicts

	2. Configuration Options: * Configurable loading priorities * Adjustable timing for non-critical surveys * Display triggers based on page performance metrics * Bandwidth-aware loading options * Fallback rendering for constrained environments * Performance thresholds for automated loading decisions While specific Core Web Vitals scores vary based on implementation context, host page configuration, and network conditions, our widgets are built with performance best practices that typically result in "Good" ratings across Core Web Vitals metrics when implemented according to our guidelines. We continuously monitor and optimize performance as web standards and metrics evolve.
1.67. Do the widgets support server-side rendering?	Our widgets are primarily designed for client-side rendering, with specific architecture considerations for performance and flexibility: Current Rendering Approach: 1. Client-Side Architecture: * Widgets are implemented as JavaScript-based components * Designed for dynamic loading on the client side * Asynchronous initialization to prevent blocking page rendering * DOM injection after the host page has loaded * Event-based rendering triggered by user interactions or page events * Runtime customization based on user context and behavior 2. Implementation Method: * JavaScript snippet placed in the host page * Asynchronous loading pattern with non-blocking execution * Dynamically created iframe or shadow DOM for style isolation * API-driven configuration and content loading * Runtime decision-making for display logic and timing * Client-side personalization based on available user data Server-Side Rendering Status: 1. Current Limitations: * Native server-side rendering (SSR) is not directly supported * Widgets are designed to be loaded and rendered post-page-load * Survey content is typically fetched from our API after widget initialization * Dynamic behavior requires client-side execution

	* Personalization logic relies on browser environment information 2. Performance Optimizations Without SSR: * Minimized initial bundle size for quick loading * Deferred initialization until after critical page content loads * Resource hints (preconnect, dns-prefetch) to accelerate loading * Efficient caching strategy for widget assets * Progressive loading of survey content * Optimized critical rendering path impact Alternative Implementation Options: 1. Hybrid Rendering Approaches: * Static HTML placeholders can be server-rendered * Widget containers pre-allocated in server-rendered markup * Initial state and configuration can be embedded in the page * Client-side hydration completes the interactive elements * Content API responses can be prefetched and included in initial page load * Custom implementation patterns available for specific use cases 2. API-First Implementation: * Our comprehensive API allows custom server-side implementation * Survey definition endpoints can be called server-side * Custom SSR implementation possible with development resources * Template-based rendering in server environments * Data prefetching for improved performance * Framework-specific integration patterns available Future Roadmap Considerations: 1. SSR Evolution: * We're evaluating server-side rendering capabilities for future releases * Exploring hybrid rendering approaches for improved performance * Investigating static generation options for common survey types * Considering framework-specific SSR support (Next.js, Nuxt, etc.) * Performance testing SSR implementations against current approach 2. Current Best Practices: * For optimal performance with the current client-side approach: * Place widget snippets before the closing </body> tag * Use deferred or delayed loading for non-critical surveys * Implement proper resource hints in the document head * Configure appropriate triggers based on page load events * Leverage our performance configuration options

	While our widgets don't currently support native server-side rendering, our client-side implementation is highly optimized for performance and flexibility. For applications where SSR is critical, our API-first approach enables custom implementations to meet specific requirements.
1.68. Which languages do you support and are all widgets supporting multiple languages?	Surveys support full localization. Multiple languages can be configured per survey and switched via URL parameter or auto-detected via browser language. Widgets support dynamic translation switching.
1.69. How is a user identified between different platforms? For example, user votes in the app, in web chrome + in web Firefox, etc. And generates 3 votes instead of 1.	Our platform offers multiple approaches to user identification across different platforms and browsers, with varying levels of implementation complexity: User Identification Methods: 1. Explicit Identifier Passing: * Primary recommended method for consistent cross-platform identification * Unique user IDs can be passed via URL parameters (e.g., ?user_id=12345) * API implementations can include user identifiers in request payloads * Custom attributes enable association of responses with specific users * Reference IDs from CRM or other systems can be leveraged * Hashed or encrypted identifiers supported for enhanced privacy 2. Anonymous Identification Options: * Browser-specific cookies for same-browser identification * Local storage persistence for returning visitors within same browser * Device fingerprinting options (opt-in required for privacy compliance) * Session-based tracking for sequential interactions * IP-based correlation with privacy safeguards * Probabilistic matching based on response patterns and metadata Cross-Platform Identification Challenges: 1. Default Behavior: * Without explicit identifiers, responses from different platforms/browsers are treated as separate * This ensures we don't incorrectly merge feedback from different users * Each unique browser/device combination generates distinct response records * No automatic cross-device correlation without explicit identifiers * Response deduplication does not occur automatically across platforms 2. Implementation Considerations:

	* Cross-platform identification requires coordinated implementation * Consistent identifier generation and passing across touchpoints * Authentication state can be leveraged for logged-in experiences * Mobile apps require consistent identifier integration via API * Offline capabilities need sync mechanisms with identifiers Recommended Identification Strategies: 1. Authenticated User Identification: * For logged-in users, pass consistent authentication identifiers * Single sign-on systems provide reliable cross-platform identification * Account-based identifiers ensure consistency across sessions * Profile information can enrich feedback data * Historical responses can be associated with user accounts 2. Anonymous User Correlation: * For non-authenticated scenarios, implement persistent anonymous IDs * First-party cookies with appropriate duration * Generate and store consistent identifiers in your application * Pass these identifiers consistently across platforms * Implement privacy-compliant identification methods 3. Hybrid Approaches: * Recognition of both authenticated and anonymous states * Identity resolution when anonymous users authenticate * Progressive identification as more information becomes available * Response merging capabilities upon positive identification * Consent-based identifier persistence Implementation Examples: 1. Website Implementation: ```javascript // Generate or retrieve persistent user ID const userId = getUserIdentifier(); // Your implementation // Include in zenloop survey initialization zenloop.initialize({ surveyId: 'your-survey-id', userId: userId, userAttributes: { // Additional identifying information email_hash: hashEmail(userEmail), customer_reference: customerNumber }

	});