

IVF Project

Graduation Project Documentation

Data Engineering Track

ONL3\_AIS5\_G1

December 2025

**IVF Project**

**BY**

**Rawan Arafa Abdelmaksoud Fatma Mohamed**

**Dina Nabil Ahmed Gehad Abbas Taha**

**Menna Abdelnasser Galal Sherri Sami Morris**

**Tasneem Tarek Abdulrahim**

**Supervisor**

**Eng/ Ahmed Hekal**

**Project Planning & Management**

1. **Project Proposal**
   1. **Overview:**

Elysium is a data-driven clinical decision-support system designed for In-Vitro Fertilization (IVF) clinics. It transforms historical and operational fertility clinic data into structured, meaningful insights to assist medical staff and administrators in making evidence-based clinical decisions.

The system consolidates clinical records, hormonal and laboratory results, and IVF cycle data into a unified structured database. Through a well-defined data engineering pipeline, it enables efficient extraction, transformation, and loading (ELT) of data, ensuring quality and consistency.

The primary deliverable is a web-based administrative platform that provides IVF success probability estimates, periodic performance analytics, and data-driven recommendations, all while maintaining patient data confidentiality and regulatory compliance.

* 1. **Objectives:**

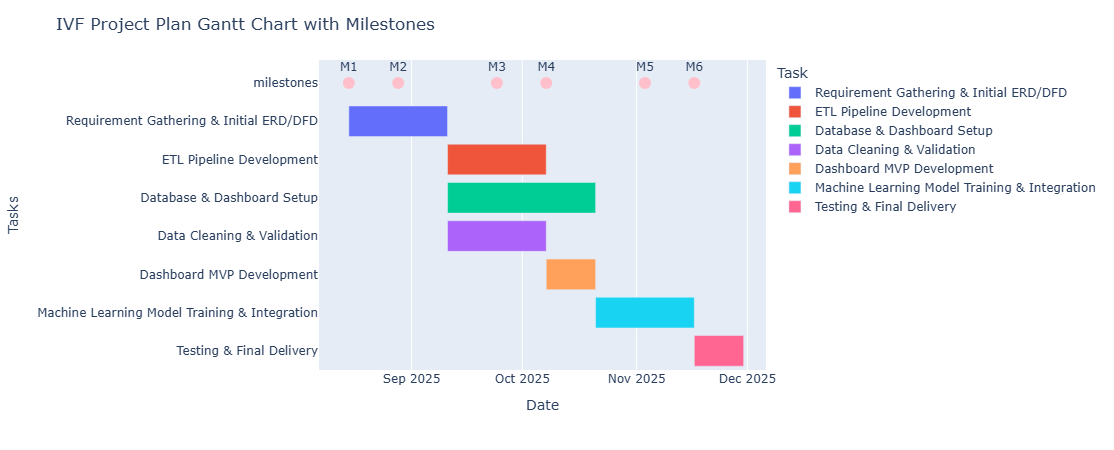
The main objectives of the Elysium system are to:

* **Develop** a predictive model to estimate IVF cycle success probability based on key clinical and laboratory parameters.
* **Design** and implement a robust data engineering pipeline (ingest → clean → stage → warehouse) to maintain structured, analysis-ready data.
* **Create** an interactive dashboard for administrators and clinicians to:
  + Monitor IVF outcomes.
  + Analyze trends and influencing factors.
  + Access visualized data insights easily.
* **Provide** automated performance reports that summarize clinic statistics and model predictions.
* **Enable** a continuous learning mechanism that updates the predictive model with new outcomes to enhance accuracy and reliability.

* 1. **Scope:**
     1. **Included** (within MVP and core features):
* Ingestion of structured data sources, including clinical records, lab results, and historical IVF cycle data.
* Data cleaning and normalization of hormonal and clinical measurements.
* Feature engineering to extract key indicators for predictive modeling.
* Implementation of automated ELT pipelines and task scheduling using Apache Airflow.
* Development of an admin-facing web application for:
  + Viewing patient-level and aggregate data.
  + Filtering records and generating visual analytics.
  + Exporting performance reports.
* Applying data governance and privacy measures, including:
  + Anonymization of sensitive information.
    1. **Excluded** (out of scope for MVP / planned future work):
* Image-based embryo selection or quality assessment using computer vision.
* Real-time integration with external EMR systems (planned for later using HL7/FHIR standards).
* Mobile application for patients (only web interface for administrators in MVP).
* Genetic data integration and multi-site scalability (future enhancement phases).
* Training and evaluation of baseline machine learning models (e.g., Logistic Regression, Random Forest, XGBoost).
* Role-based access control (admin/user).
* Basic audit logging for system transparency.

1. Project Timeline
   1. **Phase 1: Data Engineering & Infrastructure Setup (Weeks 1–4)**

* **Goal:** Build a secure and reliable data foundation for the system and analysis.
* **Tasks & Milestones:**
  + Week 1: Identify data sources (patients, labs), define access roles, and set up AWS cloud structure.
  + Week 2: Build a data ingestion pipeline (Airflow + API integration) with monitoring and logging.
  + Week 3: Perform data cleaning and transformation using Pandas and PySpark, and apply data validation using Great Expectations.
  + Week 4: Create AWS Data Warehouse (Star Schema), test data accuracy and performance, and initiate project documentation in the /Documentation folder.
* **Milestone:** Functional and secure data pipeline with validated warehouse and initial documentation.  
  1. **Phase 2: MVP — Doctor Web App & Dashboard (Weeks 5–8)**
* **Goal:** Develop and connect the doctor dashboard with live data flow.
* **Tasks & Milestones:**
* Week 5: Design and implement UI/UX (soft pastel theme, intuitive layout).
* Week 6: Connect Flask backend with AWS database and enable JWT authentication and role-based access.
* Week 7: Build dashboard components (Patients, IVF Cycles, Reports, and Settings).
* Week 8: Integrate analytics charts (success rate, cycle stages, etc.) and conduct system testing.
* **Milestone:** MVP live with secure doctor access and a fully functional dashboard.
  1. **Phase 3: System Enhancement & Expansion (Weeks 9–12)**
* **Goal:** Add additional user roles and extend system functionality.
* **Tasks & Milestones:**
* Week 9: Add Clinic Administrator role (manage doctors, view reports).
* Week 10: Add Lab Staff role (input lab test results and sync with dashboard).
* Week 11: Add Patient role (view reports and receive notifications).
* Week 12: Implement notification system, finalize role-based permissions, and update documentation.
* **Milestone:** Multi-role system completed, fully tested, and documented.  
  1. **Phase 4: Finalization & Presentation (Weeks 13–14)**
* **Goal:** Deliver final product, documentation, and demo presentation.
* **Tasks & Milestones:**
* Week 13: Review and finalize all documentation, including risk assessment and system diagrams.
* Week 14: Prepare final presentation, demo video, and project submission.
* **Milestone:** Final system delivered with complete documentation and presentation.



1. **Task Assignment & Roles**

To ensure balanced workload distribution and effective collaboration, each team member was assigned both managerial and technical responsibilities according to her expertise. The managerial roles cover planning, documentation, coordination, and reporting, while the technical roles encompass the system’s full development lifecycle — from data acquisition and transformation to design, integration, and deployment.

|  |  |  |
| --- | --- | --- |
| **Member** | **Managerial Responsibilities** | **Technical Responsibilities (Tools & Technologies)** |
| **Rawan (Team Leader)** | • Prepare Project Plan (timeline, milestones, Gantt chart).  • Handle Task Assignment & Roles documentation.  • Write Risk Assessment & Mitigation Plan.  • Oversee team coordination and ensure milestone delivery.  • Approve final documentation and integration consistency. | • Define overall System Architecture and Data Pipeline Design.  • Supervise ETL processes (Python + dbt) – review and guide rather than implement everything.  • Oversee AWS RDS and S3 integration at a high level. |
| **Menna** | • Write Project Proposal (overview, objectives, scope).  • Define Technology Stack and architecture style.  • Document Software Architecture and integration flow. | • Develop Flask backend and API routes.  • Handle authentication and routing logic using Flask Blueprints.  • Perform API testing (Postman) and manage backend configurations.  • Support deployment preparation on AWS.  • Handle backend connection between Flask API and Data Warehouse. |
| **Tasneem** | • Conduct Stakeholder Analysis and gather functional requirements.  • Prepare Use Cases and User Stories. | • Develop data extraction and transformation scripts (Python + Pandas).  • Create dbt models for transformations and aggregations.  • Maintain data validation and version control.  • Support Airflow ETL validation. |
| **Dina** | • Document System Workflow Diagrams (DFD, sequence, activity).  • Design and implement Database Schema (SQL).  • Create ERD and Class Diagrams.  • Review and optimize data models for efficiency.  • Assist in system testing and validation. | • Set up AWS RDS (PostgreSQL/MySQL).  • Connect Flask ORM models to the database.  • Implement database indexing, stored procedures, and triggers for performance.  • Support ETL validation and integration with backend.  • Assist in data migration and backup processes. |
| **Fatma** | • Review UI consistency and assist in UI testing.  • Coordinate between frontend and backend integration. | • Develop frontend interfaces (HTML, CSS, JS).  • Connect UI with Flask endpoints.  • Create data visualization dashboards (Plotly/Dash or Power BI).  • Conduct usability testing and fix visualization issues. |
| **Sherry** | • Define UI/UX Guidelines (colors, layout, typography).  • Present the user interface during the final presentation. | • Design wireframes and mockups (Figma).  • Build and manage AWS Data Warehouse (Redshift/S3).  • Create data models and optimize warehouse queries.  • Collaborate with Fatma to visualize warehouse data. |
| **Jihad** | • Define KPIs and document system performance metrics.  • Review and ensure documentation consistency. | • Containerize app using Docker.  • Deploy on AWS (EC2, S3, Redshift).  • Configure CI/CD pipeline for automation.  • Conduct testing and monitoring (unit/integration). |

1. **Risk Assessment & Mitigation Plan**
   1. **Phase 1: Data Engineering & Infrastructure Setup**

|  |  |  |
| --- | --- | --- |
| **Risk** | **Description** | **Mitigation Strategy** |
| Data Integration | Issues connecting data sources (patient data, lab results). | Test early with mock data; use Airflow to automate and monitor pipelines. |
| Data Security | Patient data might be exposed due to a weak AWS setup or API leaks. | Enable encryption, apply IAM role, and review permissions regularly. |
| Data Quality | Inaccurate or missing data during extraction or cleaning. | Use validation tools (Great Expectations); log all transformations. |
| Dependency | External tools (AWS, Airflow) may fail or face downtime. | Keep local backups, document setup, and enable service alerts. |
| Performance | Slow processing as data size increases. | Optimize queries and storage, and monitor regularly. |

* 1. **Phase 2: Doctor Web App & Dashboard**

|  |  |  |
| --- | --- | --- |
| **Risk** | **Description** | **Mitigation Strategy** |
| Functional Issues | Some features (reports) may not work as intended. | Test features step by step; gather feedback from doctors. |
| UI/UX | The dashboard may not be user-friendly or clear. | Get user feedback and adjust the design before launch. |
| Data Entry Errors | Doctors might enter incorrect or incomplete data. | Add form validation; use error messages. |
| System Security | Unauthorized access to dashboards or patient data. | Apply JWT authentication and set role-based access. |
| Integration | Dashboard analytics are not updating correctly. | Check Power BI connections and schedule regular data refresh. |
| Timeline Delays | Tasks take longer due to dependency on other members. | Track tasks using Asana, adjust weekly priorities. |

* 1. **Phase 3: Predictive Modeling & Future Expansion**

|  |  |  |
| --- | --- | --- |
| **Risk** | **Description** | **Mitigation Strategy** |
| Model Accuracy | Predictive model gives incorrect IVF success predictions. | Train with diverse data and test with multiple metrics. |
| New Role Integration | Adding lab staff or patients may break existing flow. | Test permissions and schemas before deployment. |
| Notification Delay | Notifications may not appear in real-time. | Use event-based triggers and retry logic. |
| System Scalability | Website performance drops as users grow. | Implement caching, load balancing, and database scaling. |
| Maintenance | Hard to track updates as features grow. | Keep documentation, assign ownership, and review code regularly. |

**Top 5 Critical Risks & Mitigation Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risk** | **Description** | **Mitigation** |
| 1 | Data Security | Patient data leak due to poor setup. | Encrypt data, restrict access, use IAM roles. |
| 2 | Data Quality | Wrong or missing data affects reports. | Validate and clean using Airflow pipelines. |
| 3 | Functional Issues | Missing features or broken doctor workflows. | Test features before release; gather feedback. |
| 4 | Model Accuracy | Wrong IVF predictions. | Train and monitor ML models; retrain regularly. |
| 5 | Scalability | The system slows down with more users/data. | Optimize queries and infrastructure. |

1. ***Key Performance Indicators (KPIs)***

|  |  |  |
| --- | --- | --- |
| **KPI** | **Description** | **Target** |
| System Uptime | The percentage of time the system is available and running without interruption. | ≥ 99% |
| Page & Dashboard Response Time | Average time to load pages and dashboard views. | ≤ 3seconds |

* 1. ***Data Pipeline Quality***

|  |  |  |
| --- | --- | --- |
| KPI | **Description** | **Target** |
| *ETL*  *Success*  *Rate* | *Percentage of successful data extraction, transformation, and loading operations.* | *≥ 98%* |
| *Data Accuracy* | *The consistency and correctness of data between the source and the warehouse.* | *≥ 97%* |

* 1. ***User Engagement***

|  |  |  |
| --- | --- | --- |
| KPI | **Description** | **Target** |
| Active Doctor Usage | Percentage of registered doctors who use the platform weekly. | ≥ 80% |
| Usability Rating | Average score from user feedback on clarity and ease of use. | ≥ 8/10 |

* 1. ***Data Warehouse Performance***

|  |  |  |
| --- | --- | --- |
| KPI | **Description** | **Target** |
| Query Execution Time | Average time to execute analytical queries on AWS. | ≤ 5 seconds |
| Redundant Data | Percentage of duplicated data after cleaning and validation. | ≤ 10% |

* 1. ***Security & Reliability***

|  |  |  |
| --- | --- | --- |
| KPI | **Description** | **Target** |
| Unauthorized Access | Number of unauthorized access attempts during testing. | 0 |

* 1. ***Project Delivery***

|  |  |  |
| --- | --- | --- |
| KPI | **Description** | **Target** |
| Milestone Completion Rate | The percentage of project milestones completed within schedule (±1 week). | 100% |

**Lecturer Review**

**Requirements Gathering**

1. Stakeholders Analysis

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Role / Description** | **Needs & Expectations** |
| **Doctors / Fertility Specialists**  **(mvp)** | Primary users are responsible for entering patient data, managing cycles, and viewing analytics. | - Easy data entry and retrieval.- Accurate, organized patient history.- Insightful dashboard to support decisions.- Secure access to sensitive information. |
| **Patients** | End-users who can view their treatment history and receive recommendations. | - Simple, accessible interface.- Privacy and data protection.- Transparent treatment tracking.- Personalized feedback and reminders. |
| **Clinic Administrators** | Manage system operations, user permissions, and data quality. | - Smooth workflow between staff.- Role-based access control.- Data validation and monitoring tools.- Comprehensive reporting features. |
| **Data Engineers / Developers**  **(mvp)** | Technical team maintaining the data pipeline, database, and web app. | - Clear architecture and documentation.- Automated and reliable ETL processes.- Monitoring and alerting mechanisms.- Scalable cloud infrastructure. |
| **Management / Investors**  **(mvp)** | Oversee the project’s success and return on investment. | - Performance metrics and KPIs.- Clear visual insights on outcomes.- Scalability and market potential.- Compliance with healthcare data regulations. |
| **Laboratory Staff** | Input or verify hormonal and test results in the system. | - Easy upload and validation process.- Consistency between lab data and patient records.- Quick correction of errors. |

1. **User Stories & Use Cases**
2. **Functional Requirements**
   1. **Phase 1: Data Engineering & Infrastructure Setup**

* **Data Collection & Sources**
  + Import patient data from clinics (EMR systems), lab results, historical IVF cycles, and manual entries.
  + Support future addition of wearable device data.
  + Standardize data formats (CSV, SQL, API) and maintain a Data Dictionary.
* **Data Ingestion**
  + Automate extraction from databases and APIs using Python and Airflow.
  + Store raw data in a secure cloud storage (S3 / Azure Blob).
  + Schedule and monitor ETL tasks to ensure timely ingestion.
* **Data Cleaning & Transformation**
  + Remove nulls, duplicates, and inconsistent values.
  + Standardize hormonal units and categorize ages (bins).
  + Apply One-Hot Encoding and compute derived features (BMI, hormone ratios, cycle success rates).
  + Store processed data in a staging area ready for analysis.
* **Data Warehouse Design**
  + Maintain structured storage using a star schema: Fact\_IVF\_Cycles, Dim\_Patient, Dim\_Hormones, Dim\_Embryos, Dim\_Treatment.
  + Support fast analytics and querying for dashboards and ML models.
* **Data Validation & Monitoring**
  + Validate data quality using Great Expectations or custom scripts.
  + Send alerts on pipeline failures and maintain ETL logs.
  1. **Phase 2: MVP — Web Application & Dashboard**
* **Web Application Features**
  + Patient data input forms with secure authentication.
  + Doctor dashboard showing full patient history and recommendations.
  + Role-based access and responsive UI.
  + Connect frontend (HTML/CSS/JS) to backend (Flask REST API).
* **Dashboard & Visualization**
  + Display IVF success rates by age, protocol, hormones, and other key factors.
  + Interactive monthly/yearly averages and heatmaps to identify patterns.
  + Connect Power BI / Dash / Plotly to the clean database.
  + Embed the dashboard into the web app.
  1. **Phase 3: Predictive Modeling & Continuous Learning**
* **Machine Learning Pipeline**
  + Train, validate, and deploy predictive models (Logistic Regression, Random Forest, XGBoost).
  + Use MLflow to save models and track experiments.
  + Provide API endpoints for real-time predictions integrated into the web app.
* **Continuous Learning**
  + Retrain models periodically using Airflow.
  + Include new outcomes for continuous feedback and monitor model drift.
* **Mobile Integration**
  + Allow patients to submit data via mobile apps (React Native / Flutter).
  + Send reminders for tests, medications, and appointments.
* **EMR & External Integration**
  + Sync with EMR systems using HL7/FHIR standards.
  + Support automated data updates from clinics.
* **Advanced Features**
  + Integrate genetic and wearable data for advanced analysis.
  + Support multilingual interfaces (Arabic, English, French).

**Overall Functional Goals:**

* Secure, clean, and structured data pipeline.
* Interactive dashboards for doctors and patients.
* Predictive AI model with continuous learning.

1. Non-Functional Requirements
   1. **Performance**

* ETL pipelines must efficiently process large volumes of IVF-related data.
* Dashboards should load within 2–3 seconds for standard analytical queries.
* Machine learning predictions must be generated in less than 5 seconds per request.
  1. **Security**
* Implement strict role-based access control for doctors.
* Encrypt all sensitive medical data at rest and in transit using HTTPS and TLS protocols.
* Secure all API endpoints through proper authentication mechanisms.
* Maintain regular data backups and implement a disaster recovery plan.
  1. **Usability**
* Provide an intuitive and responsive web-based dashboard for seamless interaction with visual reports and analytics.
* Include input validation features in all forms.
  1. **Reliability & Availability**
* Maintain a target uptime of 99%.
* Implement automated monitoring and alerting for ETL pipeline or model failures.
* Ensure a fault-tolerant design so that failure in one module does not disrupt others.
* Maintain continuous logging and audit trails for compliance and debugging.
  1. **Scalability**
* Support increasing volumes of IVF case data.
* Enable easy integration of new machine learning models or external data sources.
  1. **Maintainability**
* Adopt a modular code architecture for ETL, ML, and dashboard components.
* Provide comprehensive documentation of data schemas and APIs.
* Include automated testing for critical functionalities.

**System Analysis & Design**

1. **Problem Statement & Objectives**
   1. Use Case Diagram & Descriptions
   2. Functional & Non-Functional Requirements
   3. Software Architecture

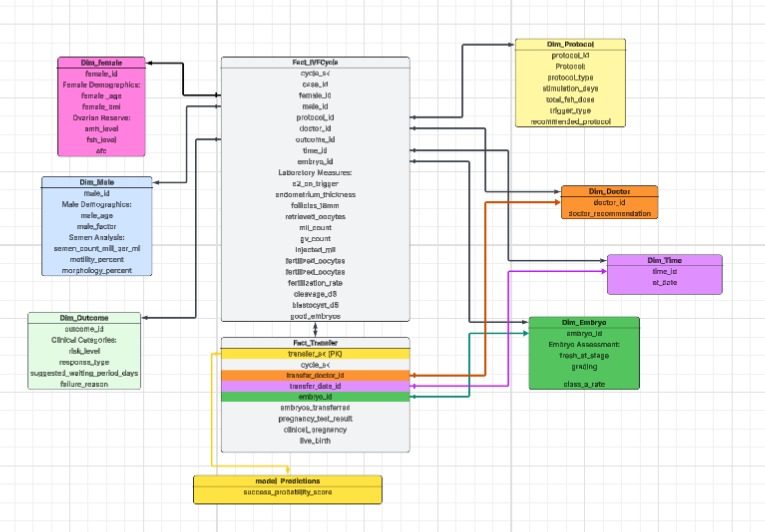
The IVF Management System is designed using the Model–View–Controller (MVC) architectural pattern, which divides the system into three main components to ensure scalability, maintainability, and a clear separation of concerns.

* **Model Layer**
  + Responsible for managing the system’s data and database interactions.
  + Stores information related to patients, IVF cycles, treatment details, and reports.
  + Handles all operations related to data storage, retrieval, and updates.
* **Controller Layer**
  + Acts as the communication bridge between the Model and View layers.
  + Receives input from the doctor (the current system user), validates it, and triggers the required logic.
  + Fetches or updates data in the Model and sends the processed results back to the View for presentation.
  + Ensures that data entered (e.g., patient details, IVF cycle updates) follows medical record standards and system validation rules.
* **View Layer (Presentation Layer**)
  + Provides the graphical user interface (GUI) through which the doctor interacts with the system.
  + Designed according to a clean, soft, and medical-friendly UI/UX concept, using:
  + Color palette: blush pink, lavender, light teal, and white.
  + Typography: rounded, modern sans-serif fonts (Poppins / Nunito).
  + Mood: calm, caring, and trustworthy.
  + Ensures a responsive layout with a sidebar for navigation and dashboard visualization tools.
  + Currently supports a single user role — Doctor — who manages patient registration, IVF cycles, and reports.

**System Component Interaction Overview**

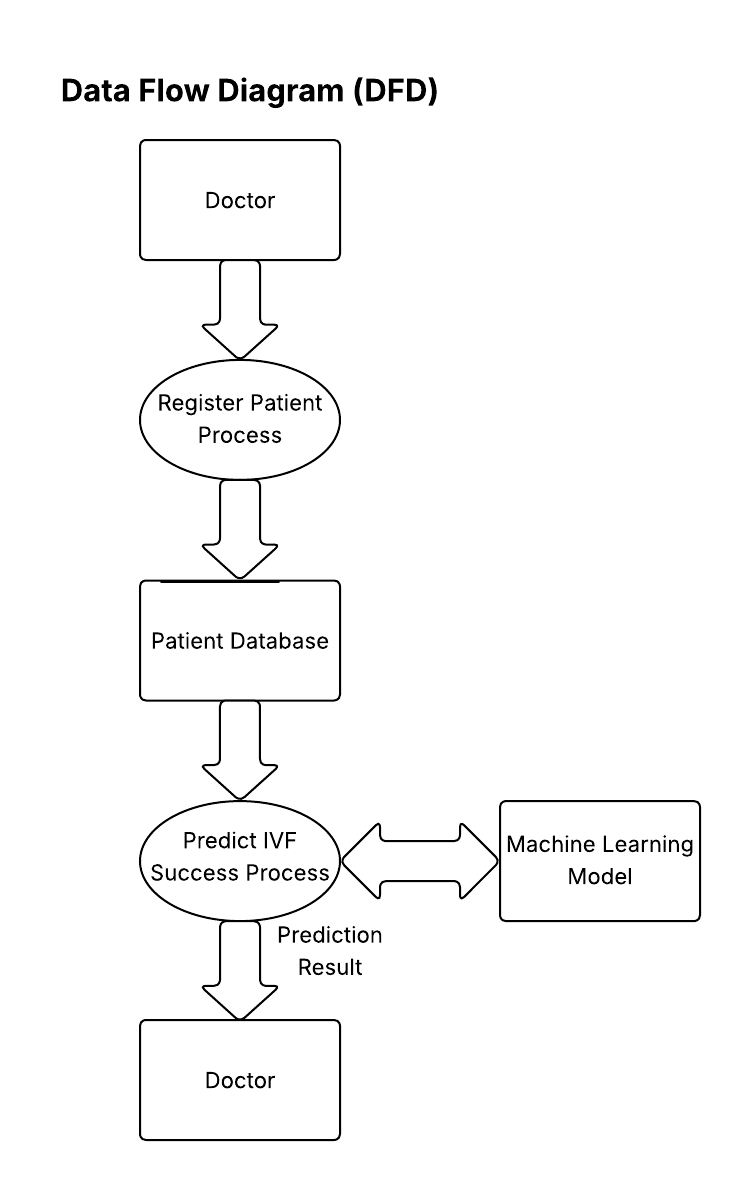
* + The Doctor (User) interacts with the View through forms and dashboards.
  + The Controller processes these requests (e.g., “Add New Patient”, “Update Cycle Details”), applies validation rules, and communicates with the Model.
  + The Model updates or retrieves the necessary data from the database.
  + The Controller sends the results back to the View, which displays updated charts, patient details, or reports.
  + This architecture ensures smooth data flow, secure input handling, and a clear modular structure that supports future scalability — such as adding more user roles (e.g., Admin, Lab Staff) or new analytical features.

1. **Database Design & Data Modeling**
   1. ER Diagram (Entity-Relationship Diagram)

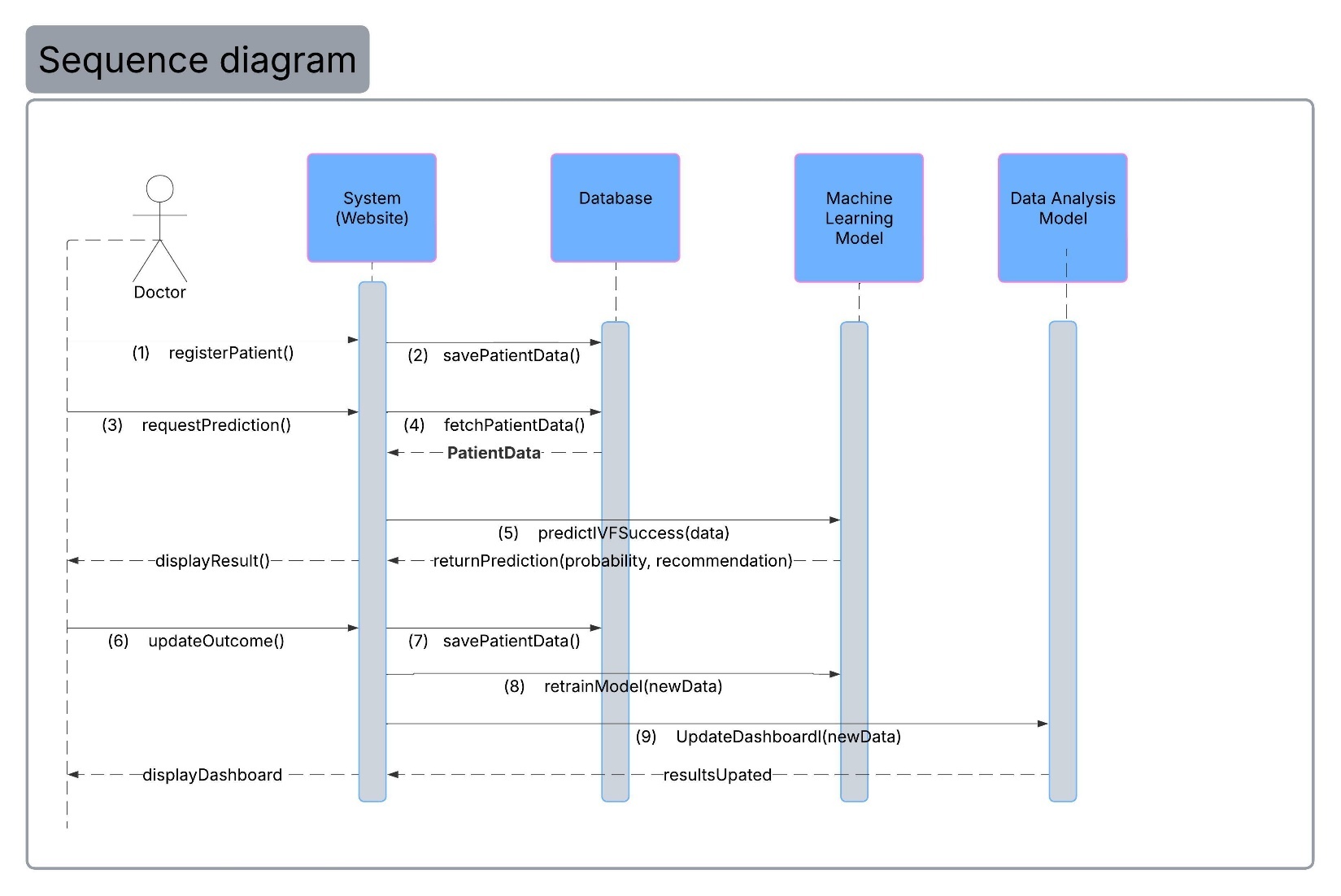


* 1. Logical & Physical Schema

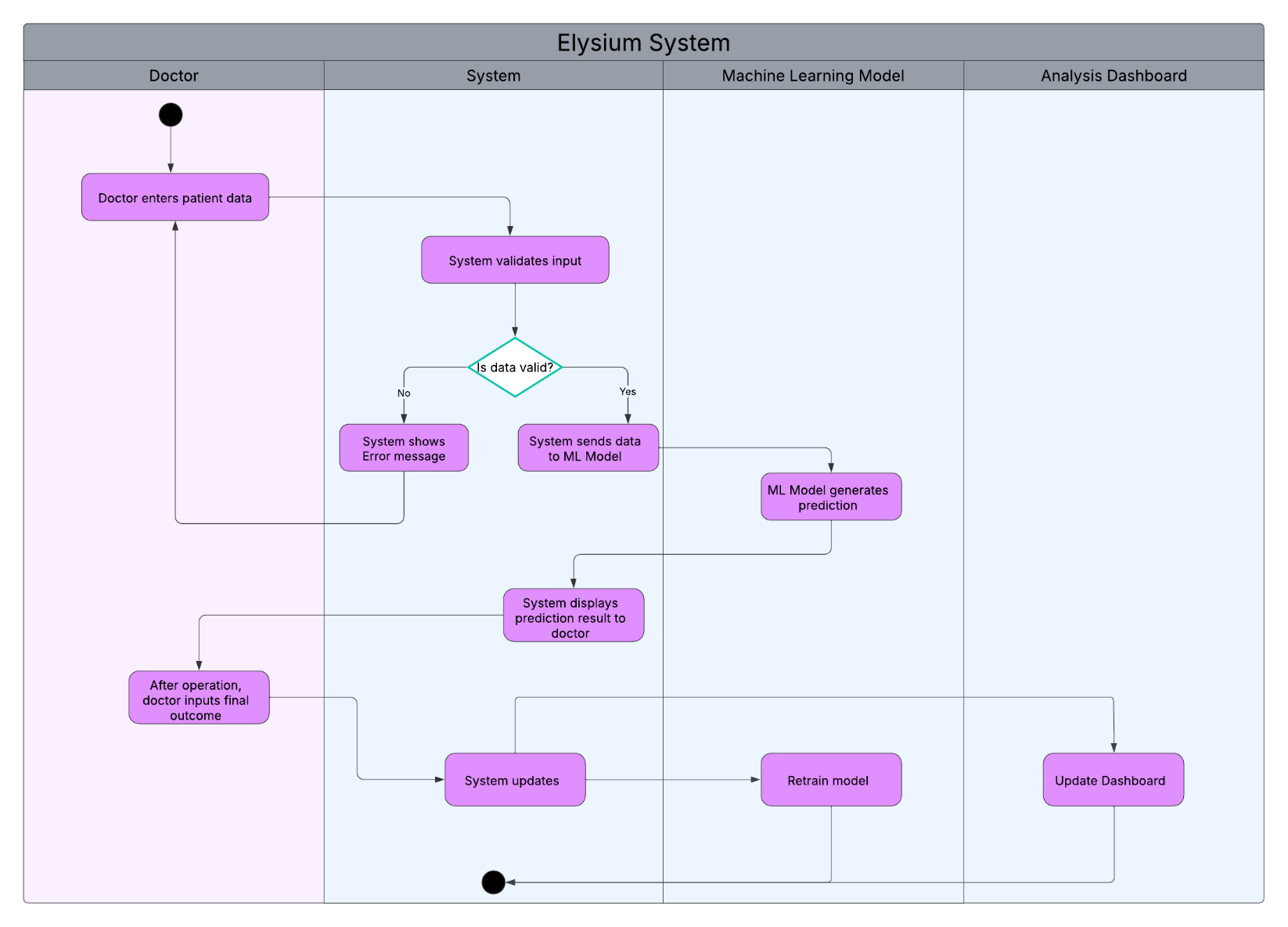
1. **Data Flow & System Behavior**
   1. DFD (Data Flow Diagram)



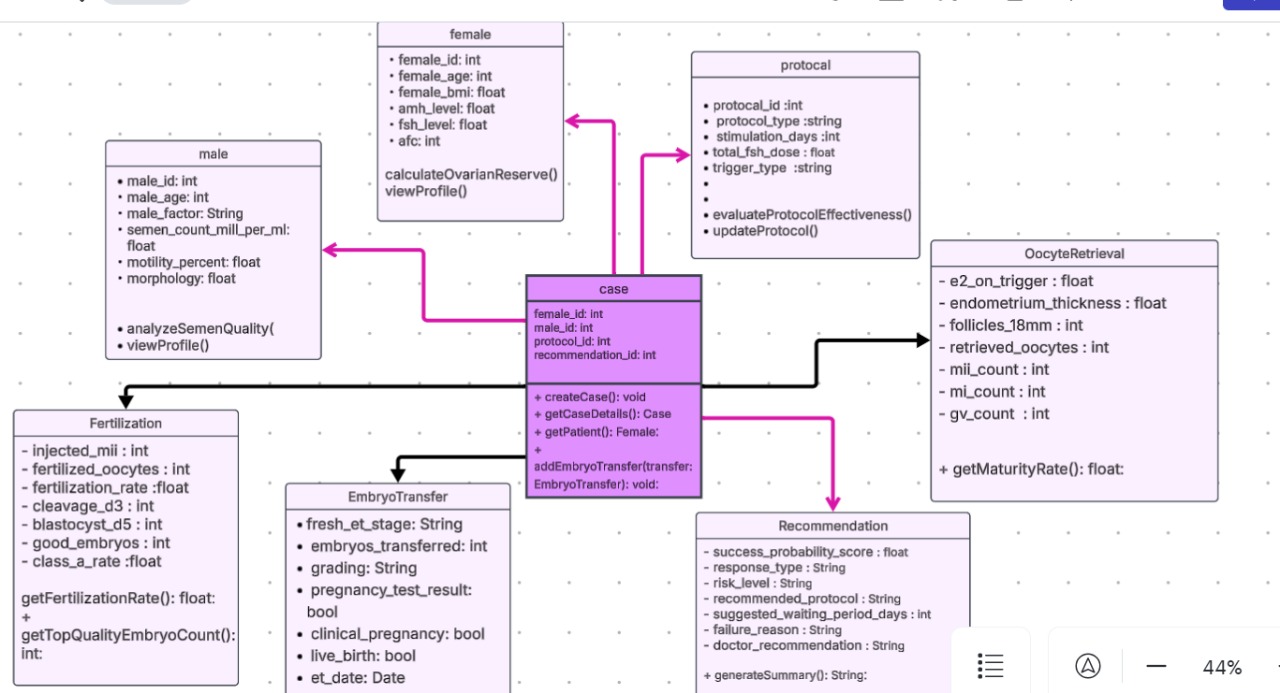
* 1. Sequence Diagrams



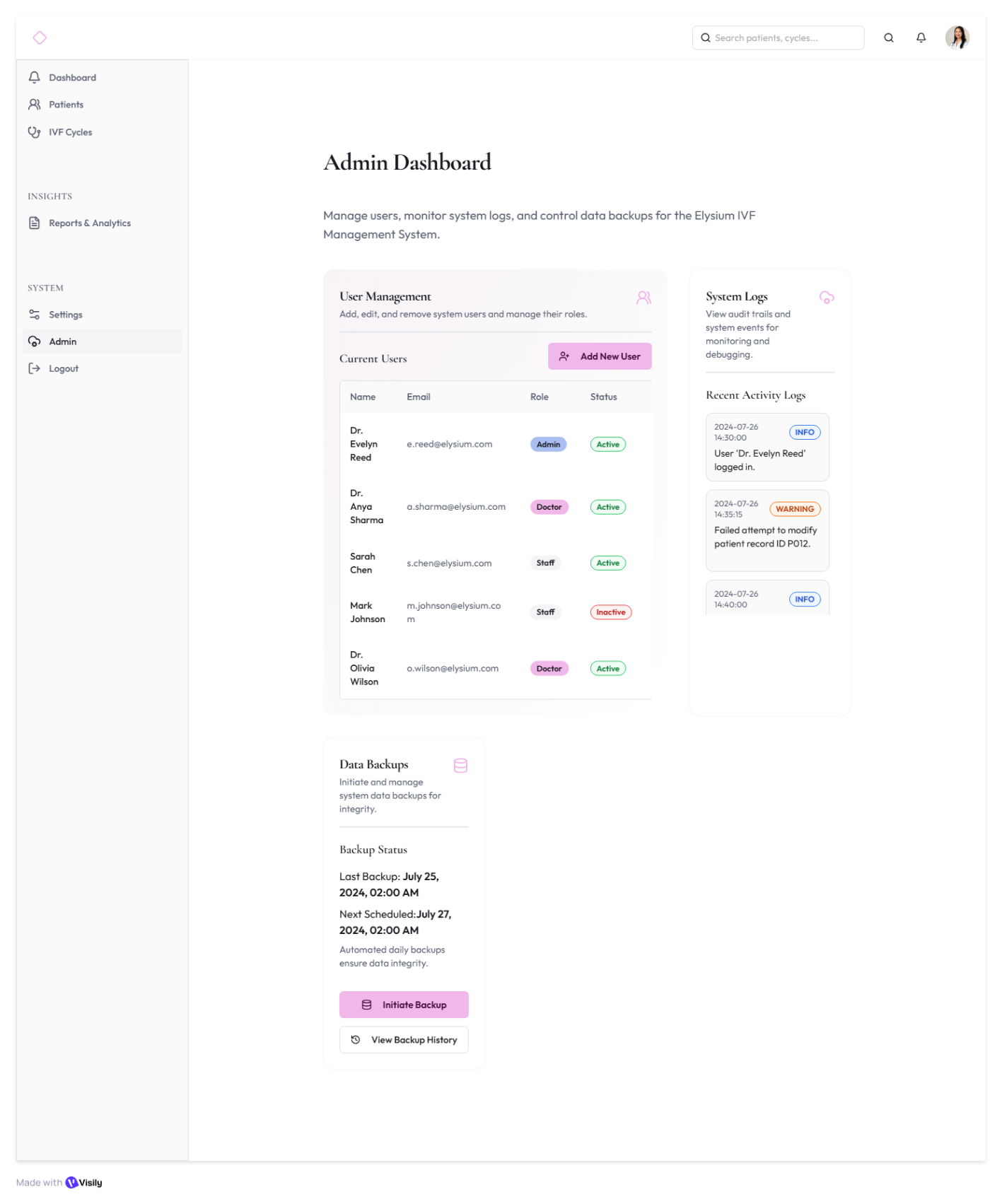
* 1. Activity Diagram

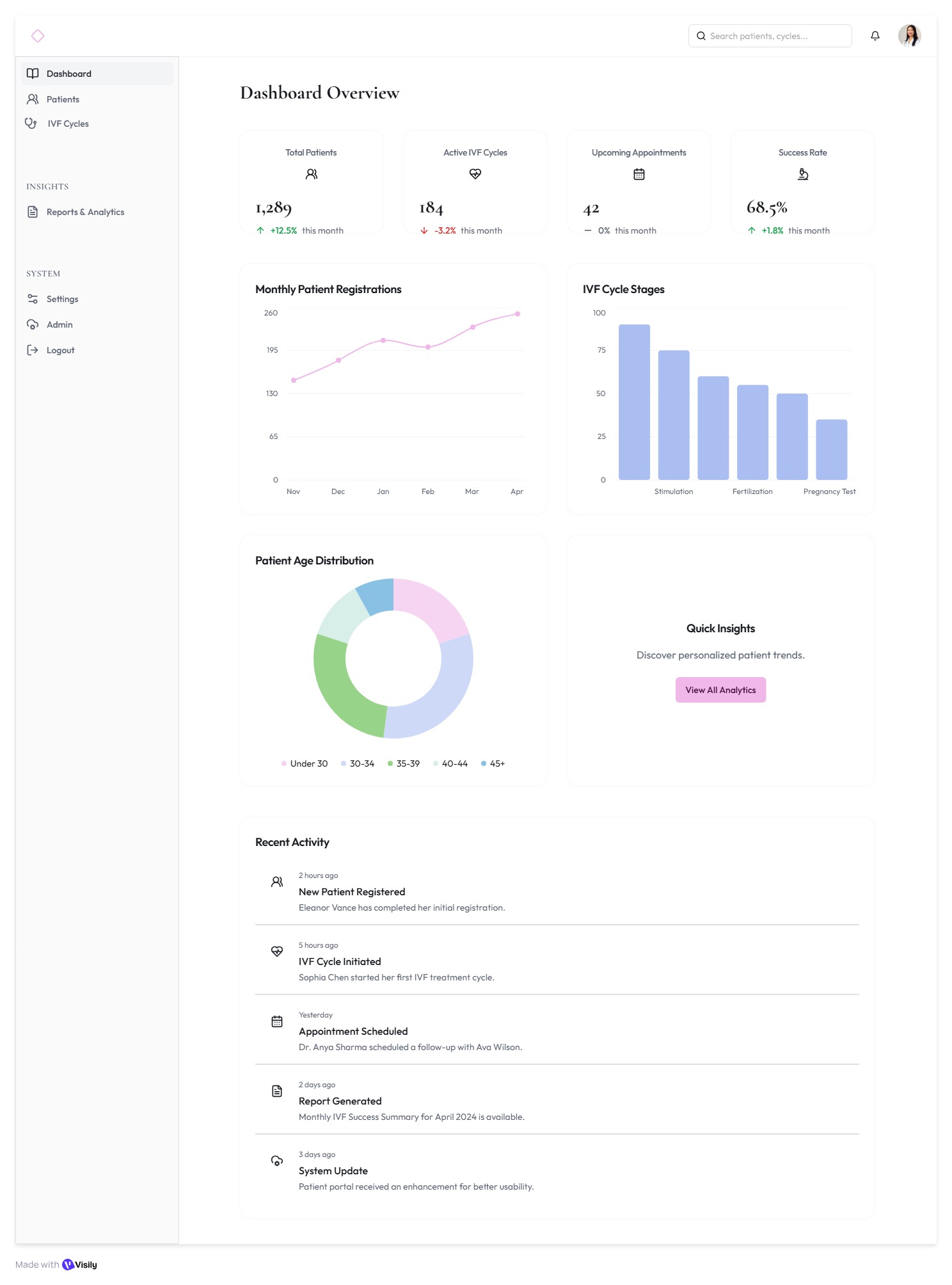


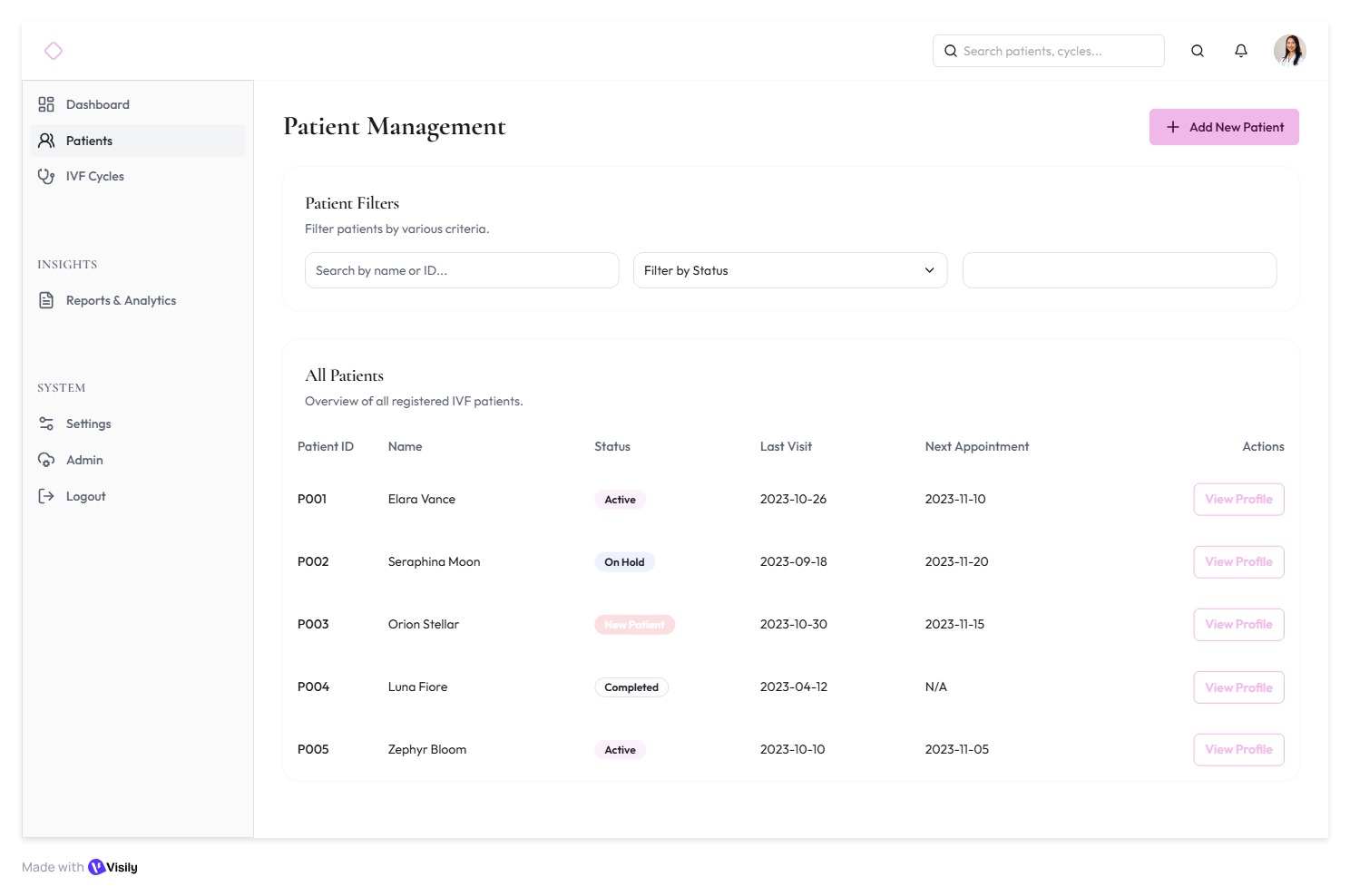
* 1. State Diagram
  2. Class Diagram

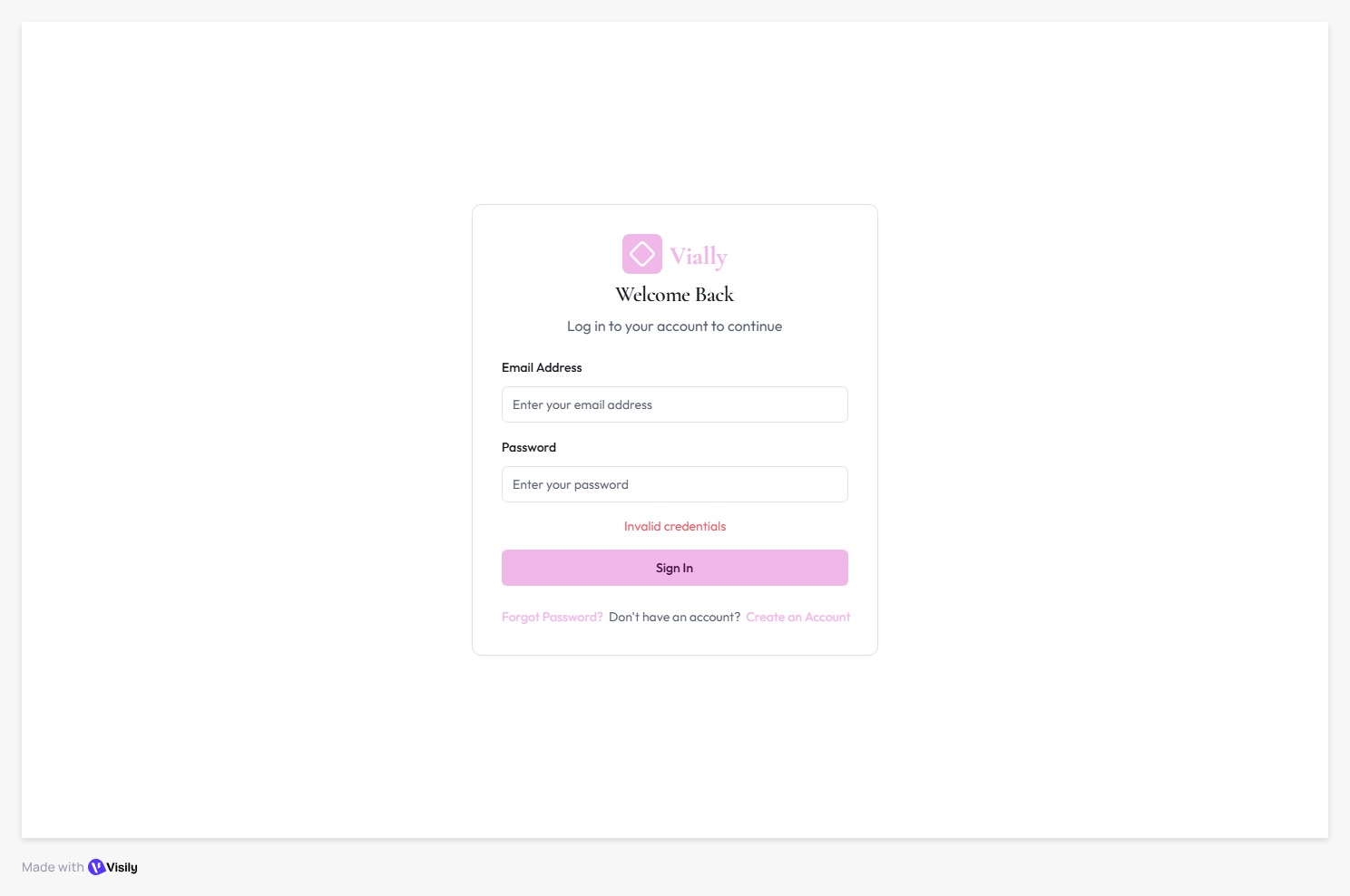


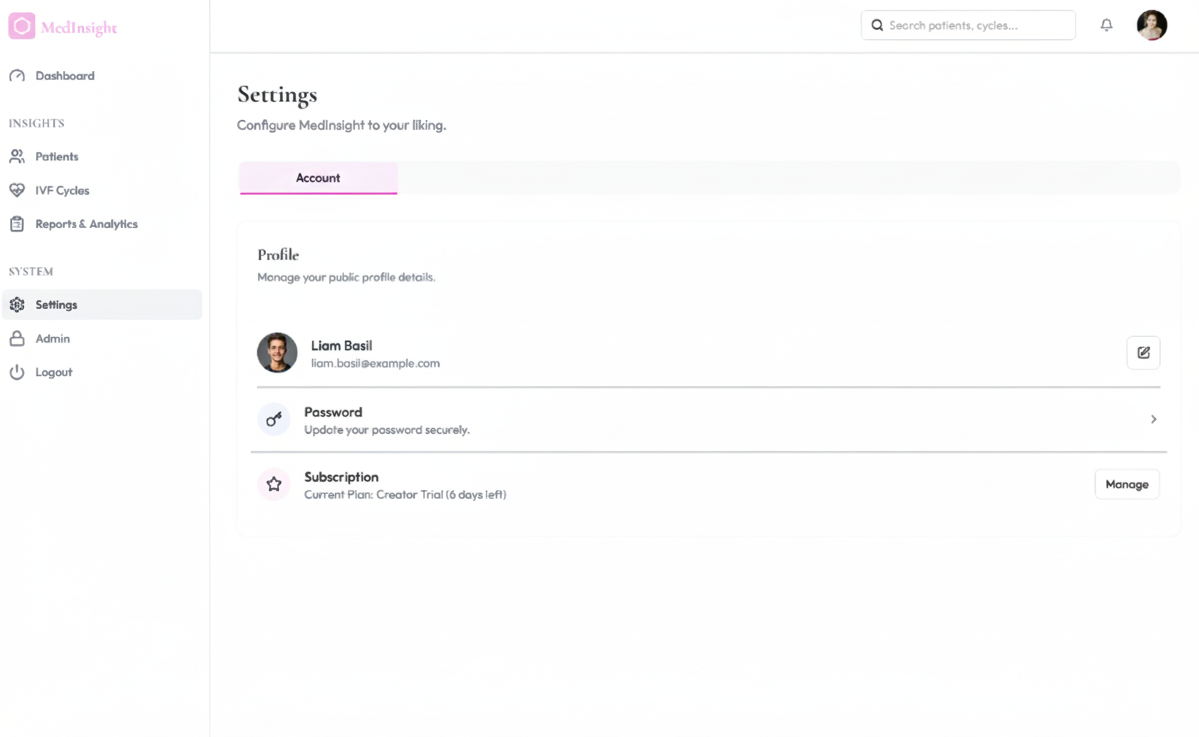
1. **UI/UX Design & Prototyping**
2. Wireframes & Mockups

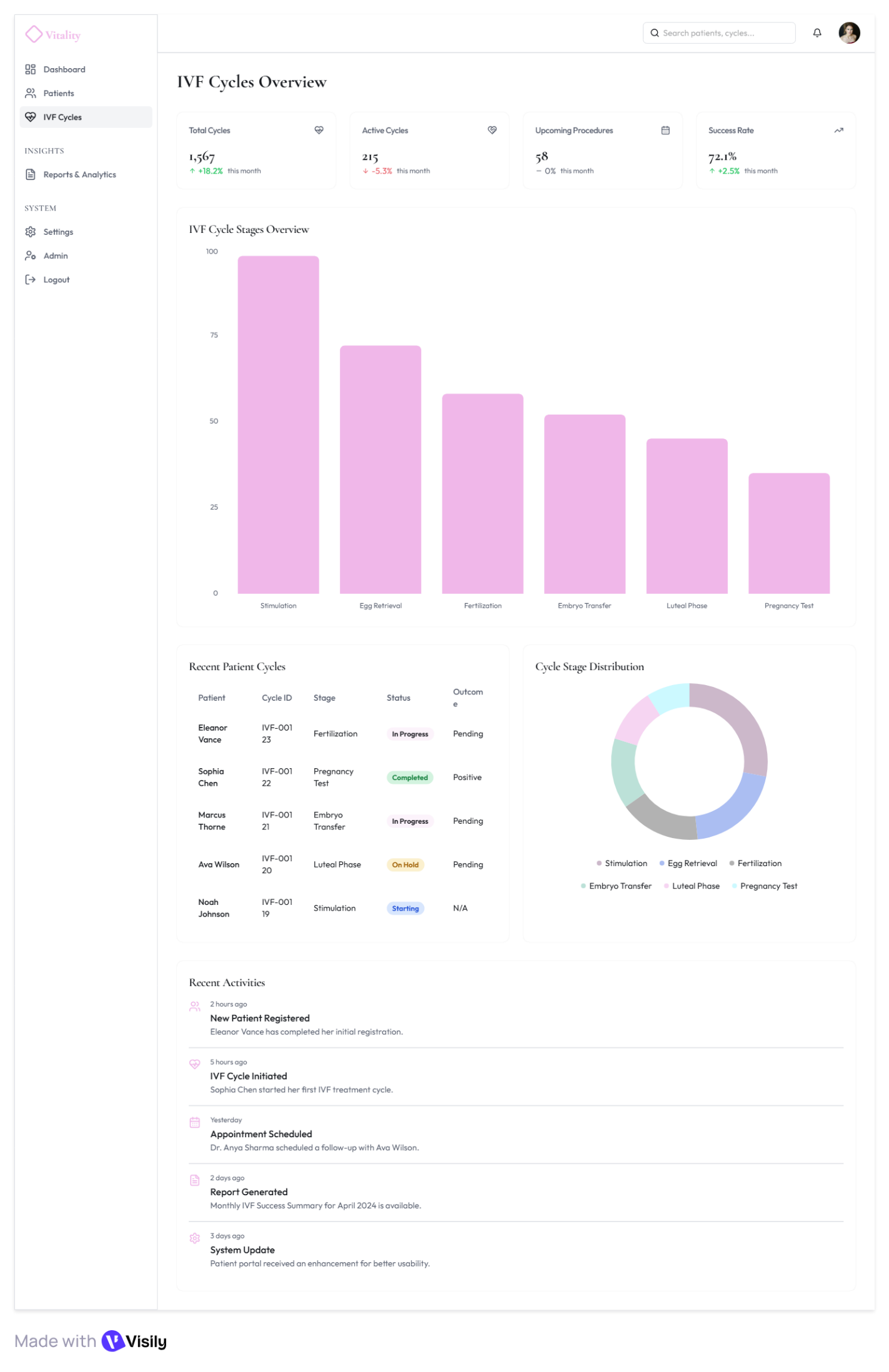


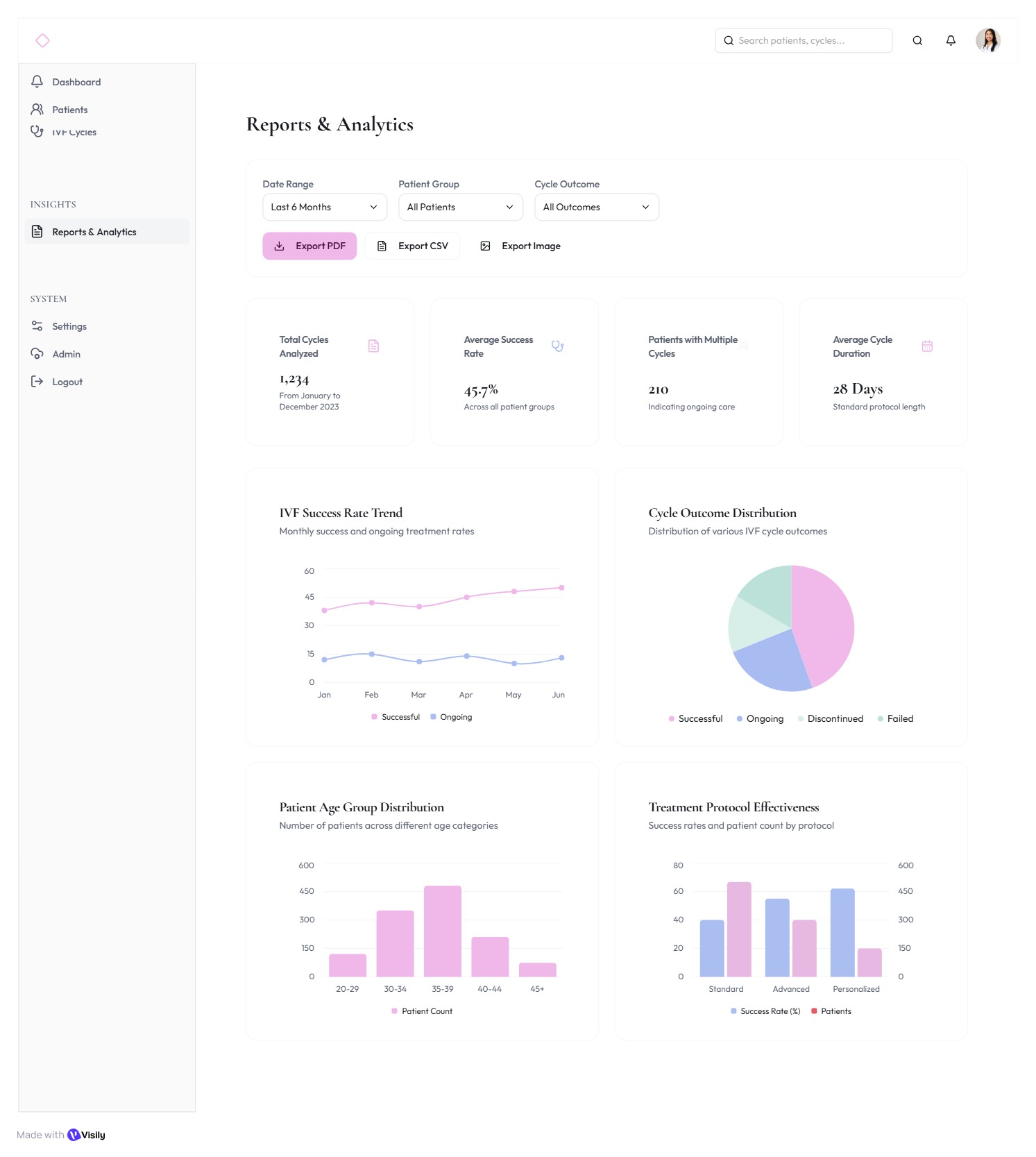
* 1. 











* 1. UI/UX Guidelines
     1. Phase 1: Doctor-Only MVP Flow
* Login / Authentication Page
* Input: Email, Password
* Validate credentials
* Redirect to Doctor Dashboard
* Error: Display "Invalid credentials" message
* Action Links: Forgot Password? and Create an Account links are available below the Sign In
* button.
* Dashboard Page

Purpose: Overview of clinic activity and insights

* + Top Cards:
  + Total Patients
  + Active IVF Cycles
  + Upcoming Appointments
  + Success Rate
* Charts & Visuals:
* Line chart for Monthly Patient Registrations
* Bar chart for IVF Cycle Stages
* Donut chart for Patient Age Distribution
* Recent Activity Feed:
* New Patient Registered
* IVF cycle initiated
* Appointment Scheduled
* Report Generated
* System Update
* Patients Page

Purpose: Manage all patient-related data

* Interface Elements:
  + Patient List Table (Patient ID, Name, Status, Last Visit, Next Appointment, Actions)
  + Search by name or ID text input
  + Filter by Status dropdown (Status options: Active, Completed, New Patient, On Hold)
  + Add New Patient (opens form)
* Patient Profile Modal (Tabs):
  + Overview
  + Medical History
  + IVF Cycle Details
  + Notes
  + Upload Reports (PDF, images)
* Action Buttons:
* View Profile (opens patient modal)
* Edit/Upload Save Changes
* Flow:

Dashboard to Patients to View Profile Action Button that opens Patient Profile Modal (Tabs

listed above) to Edit/Upload Save Changes

* IVF Cycles Page

Purpose: Track IVF cycle progress and data input

* + Sections:
  + Recent Patient Cycles
  + Cycle Stage Distribution
* Charts:
  + Total Cycles (linked to patient)
  + Active Cycles
  + Upcoming Procedures
  + Bar chart (IVF Cycle Stages Overview)
  + Donut char (Cycle Stage Distribution)
* Recent Activity Feed:
  + New Patient Registered
* IVF cycle initiated
* Appointment Scheduled
* Report Generated
* System Update
* Recent Patient Cycles section
  + Patient name id-stage-status – outcome
* Reports & Analytics Page

Purpose: Visualize clinical performance

* Filters:
  + Date range
  + Patient Group
  + Cycle Outcome
* Cards:
  + Total Cycles Analyzed
  + Average Success Rate
  + Patients with Multiple Cycles
  + Average Cycle Duration
* Visuals:
  + Line chart (IVF Success Rate Trend)
  + Pie chart (Cycle Outcome Distribution)
  + Bar chart (Patient Age Group Distribution)
  + Bar chart (Treatment Protocol Effectiveness)
  + Export as PDF/CSV/Image
* Flow:
  + Dashboard to Reports & Analytics to Filter to Export
* Settings Page

Options:

Sub-Tabs: Account

Profile Section:

 Name & Email (with Edit icon)

 Password (with Action Arrow)

 Subscription (with Manage Button)

Flow:

Dashboard  Settings  (Select Sub-Tab: Account, Display, Notifications, Integrations,

Security, Support & Danger)  Manage

7. Admin Dashboard

Purpose: Admin Dashboard - Manage users, monitor system logs, and control data backups.

Sections:

 User Management

 System Logs

 Data Backups

User management section

 User List Table columns: Name, Email, Role, Status. (This table contains all user roles:

Admin, Doctor, Staff).

 Add new user opens form for dr info

System Logs:

 View audit trails and events. Contains Recent Activity Logs with timestamps, event

descriptions, and tags (INFO, WARNING).

Data Backups:

 Shows Backup Status (Last Backup Date/Time, Next Scheduled Date/Time).

 Actions include Initiate Backup and View Backup History.

☐ Phase 2: Future Expansion (Multi-Role Access)

Added Roles & Permissions:

 Clinic Administrator - manages doctors, lab staff, patient approvals

 Lab Staff - uploads test results, updates patient labs

 Patient - views reports, progress, and messages

Notifications - triggered when:

 New patient registered

 Lab result added

 Appointment scheduled

Additional Pages:

 Appointments Page - for scheduling and tracking

 Notifications Center - unified activity log for all roles

1. **System Deployment & Integration**
   1. Technology Stack

The Elysium IVF Management System is designed with a modular and scalable architecture to ensure smooth deployment and integration across all system phases.

* + 1. **Backend (Server Layer):**
* Framework: Flask (Python) – used to build RESTful APIs for CRUD operations and data exchange between frontend and database.
* Libraries & Tools: Pandas, NumPy, SQLite3, MLflow (for model management), and Airflow (for scheduling data workflows).
* APIs: Flask REST API endpoints for patient management, IVF cycles, reports, and analytics integration.
  + 1. **Frontend (Client Layer):**
* Technologies: HTML5, CSS3, JavaScript (ES6), and Bootstrap.
* Design Style: UI/UX inspired by soft pastel color palettes (blush pink, lavender, and teal) with a rounded typography (Poppins/Nunito).
* Visualization Tools: Plotly Dash and Power BI dashboards embedded for real-time analytics and reports visualization.
  + 1. **Database Layer:**
* Database Engine: PostgreSQL (hosted on AWS RDS).
* Data Warehouse: AWS Redshift / Snowflake (for analytical workloads).
* Storage: AWS S3 used as a raw and staging data zone for ingestion pipelines.
* Schema Design: Star Schema – consisting of fact and dimension tables (Facment\_IVF\_Cycles, Dim\_Patient, Dim\_Hormones, Dim\_Embryos, Dim\_Treatt).
  + 1. **Integration & Deployment Environment:**
* Cloud Platform: AWS (S3, RDS, EC2, Redshift).
* Workflow Orchestration: Apache Airflow for automated ETL and ML pipelines.
* Monitoring & Validation: Great Expectations for data quality checks and Airflow alerts for pipeline monitoring.
* Version Control & Collaboration: GitHub.
  + 1. **Deployment Approach:**
* Flask backend deployed on AWS EC2 instance.
* Frontend served via Flask static routes.
* Database connected securely through AWS RDS.
* Airflow DAGs automated for data ingestion, transformation, and retraining pipelines.
  1. Deployment Diagram
  2. Component Diagram

