Technology Overview: Tech Triage Platform

A Non-Technical Guide to Design Decisions and Technology Choices

Last Updated: October 3, 2025

Executive Overview

The Technology Triage Platform is a web application built for Cincinnati Children's Hospital Medical Center to help reviewers evaluate invention disclosures and make informed decisions about commercialization opportunities.

What Makes This Platform Special

This isn't just a digital version of a paper form—it's an intelligent system designed to:

- **Adapt to changing needs** without requiring programmers for every update
- **Prevent errors** before they happen through built-in safety checks
- **Calculate complex scores automatically** so reviewers can focus on analysis, not arithmetic
- **Work consistently** whether accessed on a laptop, tablet, or from different locations
- **Save progress automatically** so users never lose their work

Key Benefits

For Reviewers:

- Professional, easy-to-use interface
- Automatic calculations eliminate manual math errors
- Progress is saved continuously
- Clear guidance at every step

For Administrators:

- Change form questions without technical expertise using the form builder
- Generate reports and export data easily
- Track submissions and analytics
- Deploy updates without downtime

For the Organization:

- Reliable, professional-grade technology
- Built for the long term with modern, maintainable tools
- Easy to enhance and expand as needs evolve
- Runs securely in the cloud with automatic backups

The Big Picture: How It Works

The Simple Explanation

Think of this platform as a sophisticated digital assistant that:

1. **Presents the right questions** at the right time
2. **Remembers your answers** as you work
3. **Does the math** automatically using predefined scoring formulas
4. **Adapts the experience** based on your responses
5. **Creates professional reports** from your submissions

The Smart Design: Database-Driven Forms

The platform uses a **database-driven approach** where all form questions,

sections, and validation rules are stored in a database rather than hardcoded in the application. This means:

- **Questions and text** are stored in the database, not buried in code
- **Administrators can modify questions** using the form builder interface
- **Perfect for evolving processes** that need to adapt over time
- **No developer needed** for routine changes and updates
- **Flexible and adaptable** to changing organizational needs

This approach provides the best of both worlds: the reliability and performance of a modern web application with the flexibility to adapt without requiring software developers for every change.

Technology Choices: The "Why" Behind Each Decision

Every technology choice in this platform was made to solve a specific problem or provide a particular benefit. Here's what we chose and why:

Next.js: The Modern Web Foundation

What It Is:
A framework for building web applications that work smoothly and load quickly.

Why We Chose It:

- **Speed**: Pages load fast, even on slower connections
- **Modern Features**: Automatically handles many complex tasks behind the scenes
- **Industry Standard**: Used by major companies, ensuring long-term support
- **Developer Friendly**: Makes it easier to build and maintain the application

What This Means for You:
The platform feels responsive and professional, like using a native application rather than a clunky website.

React: The User Interface Engine

What It Is:
A system for building interactive user interfaces that update smoothly as users interact with them.

Why We Chose It:

- **Smooth Updates**: When you change a score, everything recalculates instantly without page refreshes
- **Component-Based**: Each piece of the interface (buttons, forms, tables) is reusable and consistent
- **Widely Adopted**: The most popular UI technology, with vast community support
- **Future-Proof**: Backed by Meta (Facebook), ensuring continued development

What This Means for You:
The interface responds immediately to your actions, creating a fluid, modern user experience.

TypeScript: The Safety Net

What It Is:
A programming language that adds strict rules to catch mistakes before they reach users.

Why We Chose It:

- **Catches Errors Early**: Many bugs are caught during development, not

after deployment
- **Self-Documenting**: Code explains what it does, making maintenance easier
- **Better Tools**: Developers get helpful hints and warnings as they work
- **Prevents Data Mix-ups**: Ensures data is always the type we expect (number vs. text, etc.)

What This Means for You:
Fewer bugs reach production, and the ones that do are easier to diagnose and fix quickly.

Prisma: The Database Guardian

What It Is:
A tool that acts as a smart translator between the application and the database, ensuring data is handled correctly.

Why We Chose It:

- **Type Safety**: Like TypeScript, it prevents data errors before they happen
- **Automatic Migrations**: Database structure updates happen smoothly and safely
- **Easy Queries**: Makes it simpler to retrieve and store information correctly
- **Built-in Tools**: Includes Prisma Studio, a visual browser for inspecting data

What This Means for You:
Data integrity is guaranteed—the application can't accidentally save malformed data or lose information. Every form submission is stored exactly as intended.

Real Example:
If a question expects a number (like a score from 0-3) but somehow receives text, Prisma stops it before it enters the database and alerts developers to the issue.

Zod: The Validation Expert

What It Is:
A validation library that defines rules for what valid data looks like and checks inputs against those rules.

Why We Chose It:

- **Immediate Feedback**: Users know right away if they've entered something incorrectly
- **Consistent Rules**: Validation logic is defined once and used everywhere
- **Type Integration**: Works seamlessly with TypeScript for double protection
- **Clear Error Messages**: Tells users exactly what needs to be fixed

What This Means for You:
Users can't accidentally submit incomplete or invalid forms. They get helpful, immediate feedback about what needs correction.

Real Example:
If a required field is left blank or a score exceeds the 0-3 range, Zod catches it and displays a clear message before submission.

React Hook Form: The Form Manager

What It Is:
A specialized tool for managing form data, handling user input, and coordinating validation.

Why We Chose It:

- **Performance**: Only updates the parts of the form that changed, keeping it responsive
- **Integration**: Works perfectly with Zod for validation

- **User Experience**: Doesn't slow down as forms get larger and more complex
- **Less Code**: Handles common form behaviors automatically

What This Means for You:
Even complex forms with dozens of fields feel snappy and responsive. Auto-save happens in the background without interrupting your work.

PostgreSQL: The Data Foundation

What It Is:
A robust, professional-grade database system for storing all form data, submissions, and user information.

Why We Chose It:

- **Reliability**: Battle-tested in mission-critical applications worldwide
- **Data Integrity**: Built-in protections prevent data corruption or loss
- **Powerful Queries**: Can handle complex reports and analytics efficiently
- **Scalable**: Grows with your needs without requiring replacement

What This Means for You:
Your data is safe, secure, and will remain accessible for years. The database can handle thousands of submissions without slowing down.

---

shadcn/ui: The Professional Interface

What It Is:
A collection of pre-built, professional-looking interface components (buttons, forms, dialogs, etc.).

Why We Chose It:

- **Consistent Design**: Every part of the interface follows the same design

language
- **Accessibility**: Built to work with screen readers and assistive technologies
- **Customizable**: Can be adapted to match organizational branding
- **Modern**: Follows current best practices for web interfaces

What This Means for You:
The interface looks professional, feels familiar, and works reliably across all devices and browsers.

Tailwind CSS: The Design System

What It Is:
A styling system that makes it easy to create consistent, responsive designs.

Why We Chose It:

- **Consistency**: Design tokens (colors, spacing, fonts) are defined once and used throughout
- **Responsive**: Automatically adapts to different screen sizes
- **Fast Development**: Makes it quicker to implement design changes
- **Small File Sizes**: Only includes the styles actually used, keeping the app fast

What This Means for You:
The platform looks professional on every device—desktop, tablet, or phone—with no extra effort required.

Docker: The Consistency Container

What It Is:
A technology that packages the entire application and its dependencies into a standardized container.

Why We Chose It:

- **Consistency**: Runs identically on every server, eliminating "works on my machine" problems
- **Easy Deployment**: Moving to a new server is as simple as copying a container
- **Isolation**: The application and its dependencies don't interfere with other software
- **Version Control**: Easy to roll back to previous versions if needed

What This Means for You:
Deployments are reliable and predictable. Updates happen smoothly without disrupting service.

React PDF: The Report Generator

What It Is:
A library for generating professional PDF reports directly from application data.

Why We Chose It:

- **Consistency**: Reports look identical regardless of who generates them
- **Automatic**: No manual copying of data into templates
- **Professional**: Clean, print-ready formatting
- **Flexible**: Easy to update report layouts as needs change

What This Means for You:
Generate polished PDF reports with a single click. Every report follows the same professional format.

Smart Design Decisions

Beyond choosing the right technologies, several architectural decisions make this platform particularly robust and maintainable:

1. Type Safety Everywhere

What We Did:
Implemented strict type checking at every layer—database, backend, validation, and frontend.

Why It Matters:
Entire categories of bugs become impossible. For example:

- You can't accidentally store text where a number should be
- Missing required fields are caught before submission
- Database queries can't request data that doesn't exist

The Benefit:
Fewer bugs reach production, and those that do are caught quickly with clear error messages.

---

2. Database-Driven Dynamic Forms

What We Did:
Built a complete "form engine" that reads form structure from a database rather than hardcoding it.

Why It Matters:
Non-technical administrators can:

- Add new questions without a developer
- Modify existing questions using a visual interface
- Change field types and validation rules
- Reorder sections and questions
- Create entirely new forms

The Benefit:
The platform evolves with your needs without requiring developer time for every change.

Future Vision:

Imagine creating a new evaluation form for a different department—just configure it through the form builder, no coding required.

---

3. Automatic Score Calculations

What We Did:
Replicated the Excel formulas from the original PDF form in code, with automatic recalculation as users enter data.

Why It Matters:

- **Eliminates Math Errors**: The computer does the arithmetic perfectly every time
- **Real-Time Feedback**: Users see their scores update as they work
- **Consistent Logic**: Everyone's forms are scored identically
- **Transparent**: The calculation formulas are documented and auditable

The Benefit:
Reviewers can trust the scores and focus on qualitative analysis rather than double-checking calculations.

---

4. Progressive Enhancement

What We Did:
Built the application to work smoothly even on slower connections or older devices.

Why It Matters:
Not everyone has:

- The latest computer or phone
- High-speed internet access
- Modern browsers

The Benefit:
The platform remains accessible and functional for all users, regardless of their

setup.

---

5. Comprehensive Testing Strategy

What We Did:
Implemented automated tests that verify the application works correctly after every change.

Why It Matters:
Tests catch problems before they reach users:

- **End-to-end tests**: Simulate actual user workflows
- **Type checking**: Verifies all data types are correct
- **Validation tests**: Ensures rules work as intended
- **Performance tests**: Confirms the app stays fast

The Benefit:
High confidence that updates won't break existing functionality. Problems are caught in development, not production.

---

6. Form Builder Interface

What We Did:
Created a visual interface for designing and modifying forms without touching code.

Why It Matters:
Subject matter experts can:

- Design new evaluation criteria
- Test different question formats
- Refine wording based on user feedback
- Add conditional logic (show field X only if question Y is answered a certain way)

The Benefit:
Faster iteration and improvement cycles. No waiting for developer availability to make changes.

What This Means for You

For Reviewers Using the Platform

Day-to-Day Benefits:

- Forms load quickly and respond immediately to your input
- Scores calculate automatically—no math required
- Your work is saved continuously in the background
- The interface is intuitive and requires minimal training
- Professional PDF reports are generated instantly

Long-Term Benefits:

- The interface will continue to improve based on your feedback
- New features can be added without disrupting your workflow
- Your data remains accessible and secure indefinitely

For Administrators Managing the Platform

Immediate Capabilities:

- Monitor submission statistics and trends
- Generate reports and export data
- Use the form builder to modify questions and sections
- Deploy updates without downtime
- Access data through multiple interfaces (web, database browser)

Strategic Advantages:

- Platform can evolve with organizational needs
- No vendor lock-in—all technologies are open-source or widely supported
- Clear audit trail of all changes and submissions
- Scalable to handle increasing usage over time

For the Organization

Technical Excellence:

- Built with enterprise-grade technologies used by Fortune 500 companies
- Modern, maintainable codebase that won't become obsolete
- Comprehensive documentation for future developers
- Automated testing ensures reliability

Business Value:

- Faster time-to-market for new forms and evaluations
- Reduced maintenance costs through smart design
- Lower risk of data loss or errors
- Professional appearance enhances institutional credibility

Future-Proofing:

- Can integrate with other institutional systems
- Ready for expanded use cases (other evaluation types, analytics dashboards, etc.)
- Architecture supports mobile apps if needed
- Cloud-native design enables flexible hosting options

Simple Glossary

API (Application Programming Interface)
A way for different software systems to communicate with each other. Like a menu at a restaurant—it lists what you can order and how to order it.

Component
A reusable piece of the user interface, like a button, form field, or table. Think of them as building blocks that can be arranged in different ways.

Database
A structured storage system for information, like a sophisticated filing cabinet that can quickly find and retrieve specific pieces of data.

Deployment
The process of making new versions of the application available to users. Like publishing a new edition of a book.

Framework
A foundation that provides common functionality, so developers don't have to build everything from scratch. Like using a recipe template instead of inventing every recipe from first principles.

Front-end
The part of the application users see and interact with—the interface in their web browser.

Backend
The part of the application that handles data, calculations, and business logic—the "behind the scenes" work.

Migration
A controlled update to the database structure. Like renovating a filing system while ensuring no documents are lost.

ORM (Object-Relational Mapping)
A tool (like Prisma) that translates between how applications think about data and how databases store it. Like a translator between two languages.

Responsive Design
Interfaces that automatically adapt to different screen sizes, from phones to large monitors.

Type Safety
Programming language features that prevent data from being used incorrectly. Like

labeled containers that only accept specific types of contents.

Validation
Checking that data meets certain rules before accepting it. Like proofreading for correctness before publishing.

Container (Docker)
A standardized package containing an application and everything it needs to run, ensuring consistency across different environments.

Conclusion

The Technology Triage Platform represents a thoughtful balance of proven technologies and smart design decisions. Every choice—from Next.js to Docker to the database-driven architecture—was made to deliver reliability, flexibility, and maintainability.

The result is a platform that:

- **Works reliably** with modern, battle-tested technologies
- **Adapts flexibly** to changing needs through the form builder
- **Prevents errors** through comprehensive type safety and validation
- **Performs efficiently** even under heavy use
- **Scales gracefully** as organizational needs grow

Most importantly, it's built on a foundation of widely-adopted, well-supported technologies that will remain relevant and maintainable for years to come.

For more detailed technical information, see `README.md` and `CLAUDE.md` in the project repository.