Software Design Document

By G.A.I.N.S.

Table Of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to present a detailed design for the GAINS web application being developed for the CSC 230 Software Development course. This document refines and extends the previously created requirements specification by describing how the system will be structured and how its components will interact. It will serve as a guide for implementation, testing, and future maintenance.

The GAINS system is designed to provide social science students with an accessible, browser-based platform for running R statistical analyses. Students can select from pre-built statistical tools, input or upload their own datasets, generate R code automatically, view results (tables, text, plots), and download outputs for use in assignments or research. The platform also exposes the underlying R code to help students learn how statistical analyses are performed.

1.2 Scope

The GAINS application will be a full-stack web system with the following scope:

Frontend: Built with Next.js and React to provide an intuitive, responsive, and ADA-compliant user interface for selecting statistical tools, inputting data, and retrieving results.

Backend: Implemented using Node.js with Express to handle user authentication, manage statistical tool configurations, execute R code through Rscript, and return outputs to the frontend.

Database: MongoDB for storing user accounts, statistical tool templates, datasets, analysis history, and output references.

Key features in scope:

- Interactive homepage with "R Made Easy" branding
- User registration and authentication system
- Pre-built statistical tools (Linear Regression, Bar Charts, Line Charts)
- Interactive data input and visualization
- Automatic R code generation based on user inputs
- Copy and download functionality for generated R code
- Responsive design for desktop and mobile devices

Non-functional scope includes reliability, security, responsive performance, concurrent user support, and accessibility through common web browsers.

1.3 Definitions and Abbreviations

Term / Abbreviation	Definition
R	Programming language for statistical computing and graphics
Rscript	Command-line utility used to execute R scripts from the backend server
Express	Minimalist web framework for Node.js used to build APIs and handle HTTP requests
Next.js	React framework for building full-stack web applications
Frontend	Client-side component of the application (browser)
Backend	Server-side component handling logic and communication with the database
Statistical Tool	Pre-configured analysis template (e.g., Linear Regression, Bar Chart)
GAINS	Project acronym for the R statistical analysis platform
SPA	Single Page Application
API	Application Programming Interface
UI/UX	User Interface/User Experience

1.4 References

- CSC 230 Project – Requirements Final Document
- GAINS GitHub Repository: https://github.com/BradenJung/CSC230-Software-Design-and-Engineering-GAINS
- GAINS Discord Server: https://discord.gg/H3NGTmKs
- R Project for Statistical Computing: https://www.r-project.org/
- Next.js Official Documentation: https://nextjs.org/docs
- MongoDB Documentation: https://www.mongodb.com/docs/

- Node.js & Express Documentation: https://nodejs.org/en/docs

2. System Overview

2.1 System Architecture

The GAINS application follows a three-tier architecture pattern:

1. **Presentation Tier (Frontend)**: Next.js/React application serving the user interface
2. **Logic Tier (Backend)**: Node.js/Express server handling business logic and API endpoints
3. **Data Tier (Database)**: MongoDB storing persistent data

The system is designed as a Single Page Application (SPA) with server-side rendering capabilities provided by Next.js.

2.2 System Context

The GAINS system operates within the following context:

Primary Users: Social science students and researchers who need to perform statistical analyses but may have limited R programming experience.

External Systems:

- R Statistical Computing Environment (for code execution)
- Web browsers (Chrome, Firefox, Safari, Edge)
- File system (for data uploads and downloads)

Operating Environment:

- Client: Modern web browsers with JavaScript enabled
- Server: Node.js runtime environment
- Database: MongoDB instance (local or cloud-hosted)

2.3 User Classes and Characteristics

Student Users:

- Primary user class
- Limited to moderate R programming experience

- Need guided interface for statistical analysis
- Require educational features showing generated R code

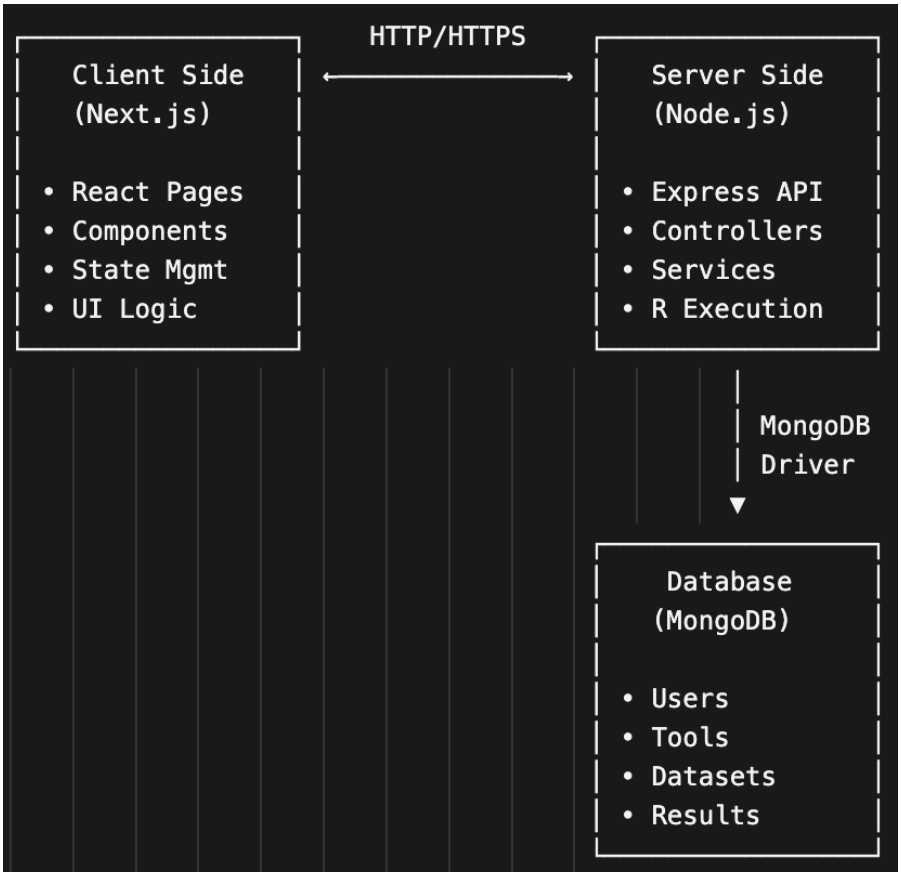
Instructor Users (Future Enhancement):

- Secondary user class
- Advanced R programming knowledge
- May need administrative features for managing student accounts

3. System Architecture

3.1 Architectural Design

The GAINS system implements a **Client-Server Architecture** with the following key characteristics:



3.2 Component Overview

Frontend Components:

- **Homepage**: Landing page with "R Made Easy" branding
- **Header/Footer**: Navigation and branding components
- **Authentication**: Login/Signup pages
- **Tool Dashboard**: Interface for selecting statistical tools
- **Data Input**: Forms for manual data entry and file uploads
- **Results Display**: Visualization of analysis outputs
- **Code Viewer**: Display and download generated R code

Backend Components:

- **Authentication Controller**: User registration and login
- **Tool Controller**: Manages statistical tool configurations
- **Data Controller**: Handles dataset uploads and processing
- **Analysis Controller**: Executes R scripts and returns results
- **User Service**: User management business logic
- **R Execution Service**: Interface to R statistical environment

3.3 Technology Stack

Frontend Technologies:

- **Next.js 15.5.2**: React framework for server-side rendering
- **React 19.1.0**: Component-based UI library
- **CSS Modules**: Scoped styling system
- **JavaScript ES6+**: Modern JavaScript features

Backend Technologies:

- **Node.js**: JavaScript runtime for server-side development
- **Express.js**: Web application framework
- **MongoDB**: NoSQL database for data persistence
- **Mongoose**: MongoDB object modeling library

Development Tools:

- **npm**: Package manager
- **Git**: Version control system
- **VS Code**: Development environment

4. Data Design

4.1 Database Design

The MongoDB database consists of the following collections:

Users Collection

```
{
  _id: ObjectId,
  username: String (required, unique),
  email: String (required, unique),
  password: String (hashed),
  firstName: String,
  lastName: String,
  createdAt: Date,
  lastLogin: Date,
  analysisHistory: [ObjectId] // References to Analysis documents
}
```

Statistical Tools Collection

```
{
  _id: ObjectId,
  toolId: String (unique), // e.g., "linear-regression"
  name: String, // e.g., "Linear Regression"
  description: String,
  category: String, // e.g., "regression", "visualization"
  icon: String,
  color: String,
  rCodeTemplate: String, // R code template with placeholders
  parameters: [{
    name: String,
    type: String, // "numeric", "text", "data", "boolean"
    required: Boolean,
    defaultValue: Mixed,
    validation: Object
  }],
  sampleData: [Object],
  createdAt: Date,
```

```
  updatedAt: Date
}
```

Datasets Collection

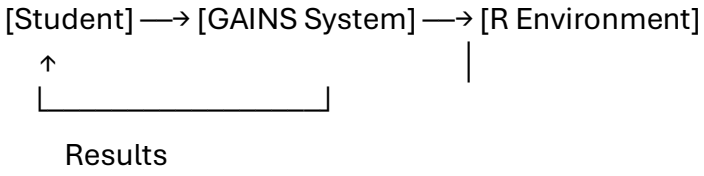
```
{
  _id: ObjectId,
  userId: ObjectId, // Reference to Users
  name: String,
  description: String,
  data: Mixed, // JSON representation of dataset
  format: String, // "csv", "json", "manual"
  columns: [String],
  rowCount: Number,
  createdAt: Date
}
```

Analysis Results Collection

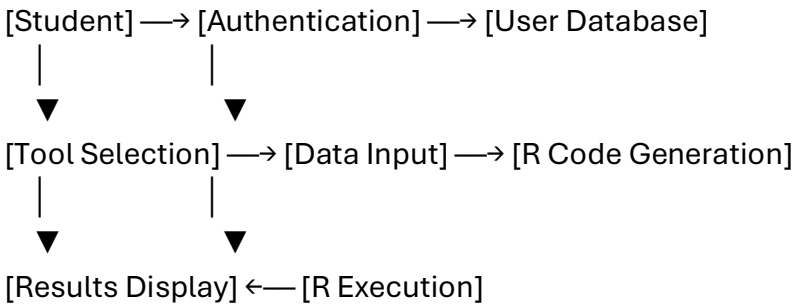
```
{
  _id: ObjectId,
  userId: ObjectId, // Reference to Users
  toolId: String, // Reference to Statistical Tools
  parameters: Object, // Input parameters used
  generatedCode: String, // Generated R code
  results: {
    output: String, // Text output from R
    plots: [String], // Base64 encoded plot images
    tables: [Object], // Tabular results
    errors: [String] // Any error messages
  },
  executionTime: Number, // Milliseconds
  createdAt: Date
}
```

4.2 Data Flow Diagrams

Level 0 - Context Diagram



Level 1 - System Process Diagram



4.3 Data Dictionary

Entity	Attribute	Type	Description
User	username	String	Unique identifier for user login
User	email	String	User's email address for communication
User	password	String	Hashed password for authentication
Tool	toolId	String	Unique identifier for statistical tool
Tool	rCodeTemplate	String	R code template with parameter placeholders
Dataset	data	JSON	User's dataset in JSON format
Analysis	generatedCode	String	Complete R code generated from template
Analysis	results	Object	Output from R script execution

5. Component Design

5.1 Frontend Components

Homepage Component (pages/index.js)

Purpose: Landing page with branding and navigation **Responsibilities**:

- Display "R Made Easy" title with gradient effects
- Provide login/signup navigation
- Showcase application features
- Responsive design for all devices

Props: None **State**: None **Methods**: None

Header Component (components/header.js)

Purpose: Navigation header for all pages **Responsibilities**:

- Display navigation links
- Show user authentication status
- Provide access to main features

Props:

- user: Current logged-in user object
- onLogout: Function to handle user logout

Tool Dashboard Component (pages/linear-regression.js)

Purpose: Main interface for statistical analysis **Responsibilities**:

- Display available statistical tools
- Show tool details and parameters
- Handle data input and visualization
- Generate and display R code
- Provide copy/download functionality

State:

- selectedTool: Currently selected statistical tool
- toolData: Configuration for selected tool
- userInputs: User-provided parameters and data

Methods:

- handleToolSelection(toolId): Switch between tools
- updateParameters(params): Update tool parameters
- generateCode(): Create R code from inputs
- executeAnalysis(): Send request to backend for execution

5.2 Backend Components

Authentication Controller (controllers/authController.js)

Purpose: Handle user authentication operations **Responsibilities**:

- User registration with validation
- User login with password verification
- JWT token generation and validation
- Password hashing and security

Methods:

- register(req, res): Create new user account
- login(req, res): Authenticate user credentials
- logout(req, res): Invalidate user session
- validateToken(req, res, next): Middleware for protected routes

Analysis Controller (controllers/analysisController.js)

Purpose: Handle statistical analysis operations **Responsibilities**:

- Process analysis requests
- Execute R scripts
- Return formatted results
- Store analysis history

Methods:

- executeAnalysis(req, res): Run statistical analysis
- getAnalysisHistory(req, res): Retrieve user's past analyses

- generateCode(req, res): Generate R code without execution

R Execution Service (services/rExecutionService.js)

Purpose: Interface with R statistical environment **Responsibilities**:

- Execute R scripts safely
- Handle R output parsing
- Manage temporary files
- Error handling and logging

Methods:

- executeRScript(code, data): Run R code with data
- parseResults(output): Process R output into structured format
- handleErrors(error): Format R errors for frontend display

5.3 API Design

Authentication Endpoints

```
POST /api/auth/register
POST /api/auth/login
POST /api/auth/logout
GET  /api/auth/profile
```

Statistical Tools Endpoints

```
GET    /api/tools           # Get all available tools
GET    /api/tools/:id       # Get specific tool configuration
POST   /api/tools/:id/execute  # Execute analysis with tool
```

Data Management Endpoints

```
POST   /api/data/upload      # Upload dataset
GET    /api/data/user/:userId  # Get user's datasets
DELETE /api/data/:id        # Delete dataset
```

Analysis Endpoints

```
POST   /api/analysis/execute   # Execute R analysis
GET    /api/analysis/history   # Get analysis history
GET    /api/analysis/:id       # Get specific analysis result
```

6. User Interface Design

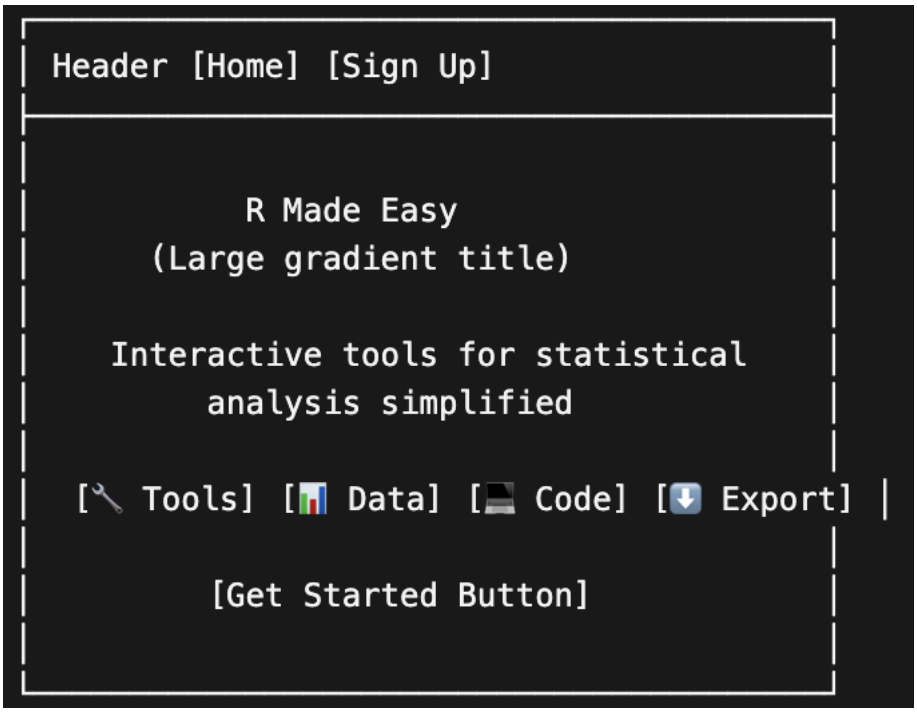
6.1 UI Architecture

The user interface follows a **component-based architecture** with the following principles:

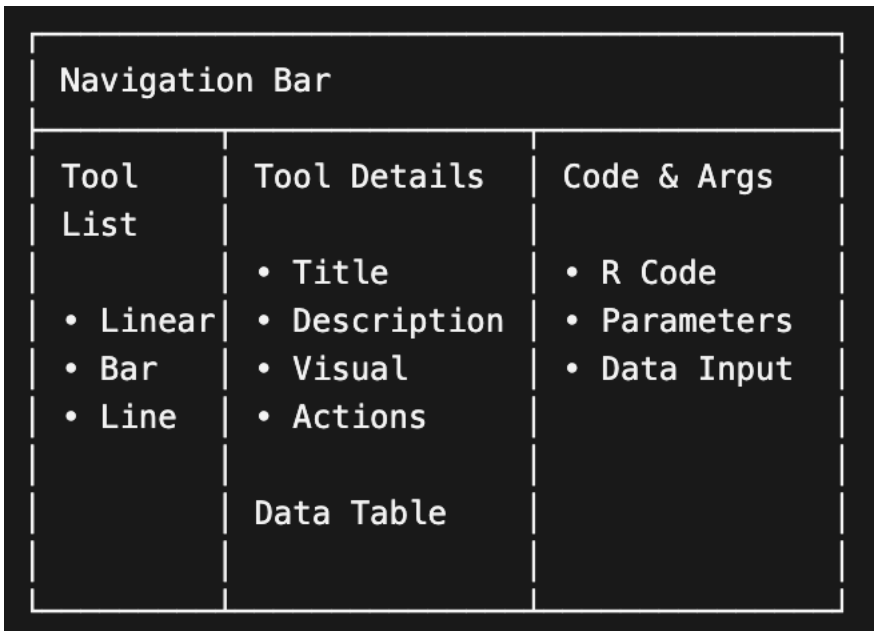
- **Responsive Design**: Mobile-first approach with breakpoints at 768px and 1200px
- **Dark Theme**: Primary dark background with neon accent colors
- **Component Reusability**: Modular components for consistent UI elements
- **Accessibility**: WCAG 2.1 AA compliance for screen readers and keyboard navigation

6.2 Screen Layouts

Homepage Layout



Tool Dashboard Layout



6.3 Navigation Design

Primary Navigation:

- Homepage → Tool Dashboard → Results
- Login/Signup flow from any page
- Breadcrumb navigation for complex workflows

Secondary Navigation:

- Tool switching within dashboard
- Parameter tabs for complex tools
- History navigation for past analyses

7. Dependencies

7.1 External Dependencies

Frontend Dependencies (from package.json):

- react: 19.1.0 - Core React library
- react-dom: 19.1.0 - React DOM rendering
- next: 15.5.2 - Next.js framework

Backend Dependencies:

- express: ^4.18.0 - Web framework
- mongoose: ^6.0.0 - MongoDB object modeling
- bcryptjs: ^2.4.0 - Password hashing
- jsonwebtoken: ^8.5.0 - JWT token handling

7.2 Internal Dependencies

Component Dependencies:

- Header component depends on authentication state
- Tool dashboard depends on tool configuration data
- Results display depends on analysis execution service

Service Dependencies:

- Authentication service depends on User model
- Analysis service depends on R execution environment
- Data service depends on file system access

Database Dependencies:

- All controllers depend on MongoDB connection
- Models depend on Mongoose schemas
- Services depend on database availability

8. Implementation

8.1 Development Environment

Required Software:

- Node.js (v18.0 or higher)
- MongoDB (v5.0 or higher)
- R (v4.0 or higher)
- Git version control
- Modern web browser for testing (ex. Chrome)

Development Setup:

1. Clone repository from GitHub
2. Install Node.js dependencies with npm install
3. Configure environment variables in .env file
4. Start MongoDB service
5. Run development server with npm run dev

Environment Variables:

```
MONGODB_URI=mongodb://localhost:27017/gains
JWT_SECRET=your_jwt_secret_key
R_PATH=/usr/local/bin/Rscript
PORT=3000
NODE_ENV=development
```

8.2 Build Process

Development Build:

- npm run dev - Start development server with hot reloading
- Automatic code compilation and browser refresh
- Source maps for debugging

Production Build:

- npm run build - Create optimized production build
- npm start - Start production server

Testing Process:

- Unit tests for individual components
- Integration tests for API endpoints
- End-to-end tests for user workflows
- R code execution tests with sample data

8.3 Deployment Strategy

Development Deployment:

- Local development environment
- Version control with Git branches

Production Deployment:

- Cloud hosting (Vercel, Netlify, or AWS)
- Environment-specific configuration (API Keys, production URLs, and settings variables)
- Database hosting (MongoDB Atlas)