Table of Contents

1.0 Login Page

1.1 Technical stack

We'll build a secure web application with the following components:

- Frontend: React.js
- Backend: Node.js with Express
- Database: PostgreSQL
- Authentication: JWT (JSON Web Tokens)
- Email Service: AWS SES

1.2 Data Dictionary

user Table

Stores user credentials, authentication status, and security-related fields.

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incremented unique user identifier
user_id	VARCHAR(50)	UNIQUE, NOT NULL	Unique username for login
email	VARCHAR(255)	UNIQUE, NOT NULL	User's registered email (used for recovery)
password_hash	VARCHAR(255)	NOT NULL	Bcrypt-hashed password
failed_login_attempts	INT	DEFAULT 0	Count of consecutive failed login attempts
account_locked	BOOLEAN	DEFAULT FALSE	If TRUE, user is locked after 3 failed attempts
last_password_change	TIMESTAMP	NOT NULL	Records when the password was last updated
password_reset_token	VARCHAR(255)	NULLABLE	JWT token for password reset (expires after 15 mins)
reset_token_expiry	TIMESTAMP	NULLABLE	Expiry time for password_reset_token
otp	VARCHAR(4)	NULLABLE	4-digit OTP for 2FA during password reset

otp_expiry	TIMESTAMP	NULLABLE	Expiry time for OTP (valid for 5 mins)
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	User account creation timestamp
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Last update timestamp

1. password_history Table

Tracks historical passwords to enforce "no reuse of last 4 passwords" policy.

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incremented record ID
user_id	INT	FOREIGN KEY (users.id)	References users.id
password_hash	VARCHAR(255)	NOT NULL	Stores hashed passwords from history
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	When the password was set

2. login_audit_log Table

Logs all login attempts (success/failure) for auditing and monitoring.

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incremented log ID
user_id	INT	FOREIGN KEY (users.id), NULLABLE	References users.id (NULL if invalid user)
ip_address	VARCHAR(45)	NOT NULL	IP address of login attempt
user_agent	TEXT	NULLABLE	Browser/device info
status	VARCHAR(20)	NOT NULL	SUCCESS, FAILED, LOCKED
attempted_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	When the attempt occurred

3. password_reset_request Table

Tracks password reset requests for security auditing.

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incremented request ID
user_id	INT	FOREIGN KEY (users.id)	References users.id
requested_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	When the reset was requested
reset_success	BOOLEAN	DEFAULT FALSE	Whether the reset was completed
ip_address	VARCHAR(45)	NOT NULL	IP address of requester

1.3 API Endpoint Specifications

Authentication API

Endpoint	Method	Parameters	Response	Status Codes
/api/auth/login	POST	{user_id, password}	{token, user_info}	200, 401, 403
/api/auth/logout	POST	{token}	{message}	200, 401
/api/auth/forgot-password	POST	{email}	{message}	200, 404
/api/auth/verify-otp	POST	{email, otp}	{is_valid}	200, 400
/api/auth/reset-password	POST	{token, new_password}	{message}	200, 400, 401

User API

Endpoint	Method	Parameters	Response	Status Codes
/api/users/change-password	POST	{current_pw, new_pw}	{message}	200, 400, 401
/api/users/password-policy	GET	-	{policy_rules}	200

1.4 Integration point

- Email Service

- **Forget Password Request**:- When a user requests to forget password, an email with a reset link/token is sent.
- **Forget User Id Request:-** When a user request forget user id an email with user id is sent

1.5 Error Handing Scenarios

Scenario	System Response	User Notification
Invalid credentials	Increment failed attempt counter	"Invalid user ID or password"
Account locked	Reject login	"Account locked for 10mins, please try again later"
Expired password	Force password change	"Your password has expired"
OTP mismatch	Reject reset	"Invalid OTP. Please try again"
Used password	Reject change	"Cannot reuse recent passwords"

2.0 Main Dashboard

2.1 Technical stack

- Frontend: React.js
- State Management: Redux Toolkit
- Data Grid: AG Grid Enterprise
- API Layer: Node.js + Express
- WebSocket: Socket.io
- Cache: Redis
- Authentication: JWT

2.2 Data Dictionary

1. business_dates Table

Tracks valid trading dates for historical lookups

Column Name	Data Type	Constraints	Description	Example
date	DATE	PRIMARY KEY	Calendar date	2023-08-21
is_trading_day	BOOLEAN	NOT NULL	Market open flag	TRUE
previous_trading_day	DATE	FOREIGN KEY (business_dates.date)	Fallback reference	2023-08-18

3. dashboard_cache Table

Stores pre-calculated Risk Engine outputs for the dashboard

Column Name	Data Type	Constraints	Description	Retention
cache_key	VARCHAR(255)	PRIMARY KEY	Composite key: accountId_date_metric	7 days
metric_value	JSONB	NOT NULL	Risk Engine API response	
expires_at	TIMESTAMPTZ	NOT NULL	TTL for cache invalidation	

2.3 API Endpoint Specifications

For Dashboard Data Fetching

Endpoint	Method	Parameters	Response Schema	Cache TTL
/api/dashboard/summary	GET	account_id, date?	{ total_margin, cleared_value, ... }	5 sec
/api/dashboard/positions	GET	account_id, filters?	{ positions: [...] }	10 sec
/api/dashboard/greeks	GET	account_id	{ delta: value, gamma: value }	5 sec
/api/dashboard/subscribe	WebSocket	account_id	Real-time updates	-

2.4 Integration Points

1. **Data Flow**:
    - Risk Engine pushes calculations → **Dashboard API** (via gRPC)
    - API transforms data → **Dashboard-optimized JSON**
2. **Key Protocols**:
    - Real-time: WebSocket (position updates)
    - Batch: REST (initial load, historical data)

3.0 What if calculation

3.1 Technology stack

- Frontend: React.js
- File Parser: Papa Pars
- Validation Engine: JOI library
- API Layer: Node.js + Express
- Database: PostgreSQL

3.2 Data Dictionary

3.3 API Endpoints

Endpoint	Method	Parameters	Request/Response Example
/api/whatif/upload	POST	account_id, file	**Request:** multipart/form-data **Response:** { scenario_id: "uuid", warnings: [...] }
/api/whatif/status	GET	scenario_id	**Response:** { status: "RUNNING", progress: 65 }
/api/whatif/results	GET	scenario_id	**Response:** { new_margin: 1350000, delta: +10%, greeks: {...} }
/api/whatif/template	GET	-	**Response:** { csv_template_url: "s3://..." }

3.4 Integration point

1. File Upload → Validation Service

Interaction Flow:

1. **Frontend** uploads CSV via POST /api/whatif/upload
2. **Backend** streams file to **Validation Service** (microservice)
3. **Validation Service** checks:
    - File structure against trade_file_templates schema
    - Mandatory fields (e.g., trade_id, ccy_pair)
    - Expiry dates within business calendar bounds

2. Job Triggering

- **Mechanism**:
  The backend publishes an event to a message queue (Kafka) containing:
    - scenario_id
    - Path to the uploaded file
    - Baseline portfolio snapshot date

- **Purpose**:
  This decouples the upload process from risk computation, ensuring the UI remains responsive.

3. Risk Engine Processing

- **Step 1**: The Risk Engine (running as a gRPC service) consumes the event from the queue.
- **Step 2**: It retrieves:
  - The **hypothetical trades** from the S3 file
  - The user's **current portfolio** from the trade database (as of the baseline date)
- **Step 3**: Executes margin calculations by:
  1. Combining the existing portfolio + new trades
  2. Applying risk models (e.g., Monte Carlo simulations for stress testing)
  3. Computing updated metrics:
     - Total margin requirement
     - Greeks exposure (delta, gamma, etc.)
     - Threshold breach status

3.5 Error Handling

Scenario	System Response	User Notification
Invalid file format	HTTP 400	"Upload CSV template v2.1 (download link)"
Missing required field	HTTP 422 + field-level errors	"Missing 'ccy_pair' in row 5"
Expired trade date	Reject upload	"trades with expired dates"
Risk engine timeout	HTTP 503 + retry button	"Calculation delayed - try again in 1 minute"

4.0 Stress Analysis

4.1 Technical Stack

- Frontend: React.js + AG Grid
- API Layer: Node.js + Express
- Cache: Redis
- Database: PostgreSQL

4.2 Data Dictionary

1. **stress_scenarios** Table

Column Name	Data Type	Description	Example
scenario_id	VARCHAR(10)	Primary key	"GFC_2008"
scenario_name	TEXT	Display name	"Global Financial Crisis"
description	TEXT	Tooltip text	"Lehman Brothers collapse effects"
is_active	BOOLEAN	If selectable	TRUE
risk_parameters	JSONB	Engine config	{"volatility_multiplier": 2.5}

2. stress_results Cache Schema (Redis)

Key Format	Data Type	TTL	Description	Example Value
stress:{account_id}:{scenario_id}	JSON	1h	Result data	{"baseline_margin": 1.2M, "stressed_margin": 1.5M, "breach_status": false}

4.3 API Endpoints

1.1 Scenario Management APIs

Endpoint	Method	Parameters	Response Example	Purpose
/api/stress/scenarios	GET	None	json [{ "scenario_id": "GFC_2008", "name": "Global Financial Crisis", ... }]	Fetch all predefined scenarios for dropdown
/api/stress/scenarios/{id}	GET	scenario_id (path)	json { "scenario_id": "GFC_2008", "parameters": { "volatility_spikes": {...} } }	Get risk parameters for a specific scenario

1.2 Portfolio Data APIs

Endpoint	Method	Parameters	Response Example	Purpose
/api/stress/portfolio/{id}	GET	account_id (path)	json { "positions": [{ "trade_id": "FX2023-056", "type": "NDF", ... }] }	Fetch current portfolio from Risk Engine

1.3 Stress Test Execution APIs

Endpoint	Method	Parameters	Response Example	Purpose
/api/stress/run	POST	{ account_id, scenario_id }	json { "result_id": "xyz", "status": "QUEUED" }	Trigger stress test job
/api/stress/status/{id}	GET	result_id (path)	json { "status": "RUNNING", "progress": 45 }	Check job status
/api/stress/results/{id}	GET	result_id (path)	json { "baseline_margin": 1200000, "stressed_margin": 1500000, "breached": true }	Fetch results (Redis)

1.4 Real-Time Updates (WebSocket)

Endpoint	Protocol	Parameters	Message Example	Purpose
/api/stress/updates	WebSocket	result_id (query)	json { "type": "PROGRESS", "data": { "progress": 60 } }	Live progress updates

4.4 Integration Point

1. Front end to Risk engine

- **User selects a stress scenario** from the dropdown, triggering a request to fetch scenario parameters via `GET /api/stress/scenarios/{id}`.
- **Risk engine retrieves portfolio data** by calling `GET /api/stress/portfolio/{account_id}`, ensuring it has the necessary trade details before stress calculations.
- **Frontend initiates the stress test** using `POST /api/stress/run`, sending the `account_id` and `scenario_id`. The risk engine queues the job and returns a `result_id`.
- **Real-time updates** are provided via WebSocket (`/api/stress/updates`), allowing the frontend to display progress (e.g., "Stress Test 50% Complete").

- **Once the test is complete**, the frontend fetches results using `GET /api/stress/results/{result_id}`. Redis temporarily stores these results for fast retrieval.

4.5 Error handling

Scenario	System Response	Notification (Frontend)
Stress selection option	Blank	"No scenarios available. Please try again later."
Portfolio data not found	404 Not Found	"No portfolio data available for this account."
Risk engine unreachable	503 Service Unavailable	"Risk engine is currently unavailable. Please retry later."
Stress test execution failed	500 Internal Server Error	"Stress test could not be executed. Please contact support."
Stress test timeout	Job stuck in QUEUED state for too long	"Stress test is taking longer than expected. Please wait or retry."
WebSocket disconnected	Connection lost	"Live updates lost. Reconnecting..."
Result expired in Redis	404 Not Found	"Test results expired. Please rerun the stress test."
User unauthorized for API	401 Unauthorized	"Session expired. Please log in again."

5.0 Report generation

5.1 Technology stack

- React.js
- File Download Libraries
- Node.js + Express
- CSV Generation Libraries
- PostgreSQL
- Redis
- JWT

5.2 API Endpoints

A. Fetching Portfolio Data APIs

Endpoint	Method	Parameters	Response	Purpose
/api/portfolio/original/{account_id}	GET	account_id (path)	json { "account_id": "ACC123", "trades": [{ "trade_id": "T001", "trade_type": "NDF",	Retrieve origina

			"ccy_pair": "EUR/USD", "base_notional": 1000000, "status": "LIVE" }, ...] }	l portfolio (live/cleared trades)
/api/portfolio/whatif/{account_id}	GET	account_id (path)	Same structure as original, with: • Added "hypothetical": true flag for new trades • Modified "base_notional", "margin_impact"	Get What-If modified portfolio
/api/portfolio/stress/{account_id}	GET	account_id (path), ?stress_period=GFC_2008 (query)	Same structure, with: • "stress_adjusted": true • "stressed_value": X replacing original values	Fetch stress-adjusted portfolio

B. Report Download API

Endpoint	Method	Parameters	Response	Purpose
/api/report/download	POST	json { "account_id": "ACC123", "report_type": "WhatIf", // Enum: "Original", "WhatIf", "Stress" "stress_period": "GFC_2008", // Required if report_type=Stress "user_id": "USER789" }	**Success (200):** • Binary CSV file with headers: Content-Disposition: attachment; filename="Portfolio_Report_WhatIf_20230821_USER789.csv" **Error (400):** json { "error": "Missing stress_period for Stress report" }	Generate and download portfolio CSV

5.3 Integration Point

Integration Flow Overview

1. Viewing the Original Portfolio:

- **User Action:** The user selects a margin account from the dashboard.
- **Frontend Integration:**
  - The React.js UI (with AG Grid for display) triggers a call to `GET /api/portfolio/original/{account_id}`.
  - The returned live and cleared trade data is displayed on the dashboard.
- **Backend Role:**
  - The Node.js/Express API fetches the original portfolio data from PostgreSQL and sends it back in JSON format.

2. Post-What-If Calculation & Report Download:

- **User Action:** After running a What-If calculation, the modified portfolio is shown on the dashboard. When the user clicks the "Download Report" button, a pop-up displays two options:
  - **Original Portfolio Report**
  - **What-If Portfolio Report**
- **Frontend Integration:**
  - Depending on the user's selection, the appropriate data endpoint is determined (either `GET /api/portfolio/original/{account_id}` for the original or `GET /api/portfolio/whatif/{account_id}` for the modified portfolio).
  - The frontend then issues a `POST /api/report/download` API call, including parameters such as the `account_id`, chosen `report_type`, and `user_id`.
- **Backend Role:**
  - The API layer receives the download request.
  - It uses a CSV generation library (e.g., **fast-csv** or **csv-writer**) to transform the portfolio JSON data into a CSV file.
  - The backend sets appropriate file headers and naming conventions (e.g., `Portfolio_Report_WhatIf_<Date>_<UserID>.csv`).

- **Libraries & Delivery:**
  - The generated CSV file is stored temporarily in Redis
  - The frontend uses file download libraries (e.g., **FileSaver.js**) to prompt the user with a download dialog.

3. Post-Stress Analysis & Report Download:

- **User Action:** Following a stress analysis, the dashboard displays stress-adjusted portfolio data.
- **Frontend Integration:**
  - A similar pop-up is shown with options for:

        - **Original Portfolio Report**
        - **Stress Analysis Portfolio Report**
    - For a stress report, the API call includes an additional parameter for the stress period.
    - The frontend calls `POST /api/report/download` with parameters like `account_id`, `report_type` set to "Stress", `stress_period`, and `user_id`.
- **Backend Role:**
    - The backend fetches stress-adjusted data from PostgreSQL or Redis (if recently calculated).
    - It uses the CSV library to generate the report, applying the naming convention (e.g., `Portfolio_Report_Stress_<StressPeriod>_<Date>_<UserID>.csv`).
    - The CSV file is then sent back to the frontend.
- **Libraries & Delivery:**
    - The frontend again leverages a file download library to prompt the file save, ensuring that the user receives the correct version of the report.

5.4 Error Handling

Scenario	System Response	Frontend Notification
No portfolio data available	404 Not Found	"No portfolio data available for this account."
Download request with missing parameters	400 Bad Request	"Required parameters are missing. Please check your selection and try again."
Report generation failure	500 Internal Server Error	"Unable to generate the report. Please try again later or contact support."
Unauthorized access to API	401 Unauthorized	"Session expired. Please log in again."
File generation timeout	504 Gateway Timeout	"The report is taking longer than expected. Please try again later."