Smart Money Chat Development Portfolio

Smart Money Chat: Empowering Financial Education Through AI-Powered Technology

Authored by Rishi Oberoi

AI-Powered Platform

Advanced chatbot technology delivering personalized financial guidance

Financial Education For All

Democratizing access to financial knowledge across all demographics

Smart Money Solutions

Data-driven insights powering informed financial decisions

SMART MONEY CHAT

Smart Money Chat | AI Finance Platform, RAG Workflow, and QA-Ready Product Surface

This page highlights my work across AI chat UX, LangChain RAG flows, Azure OpenAI integration, trusted-source retrieval, and QA-focused product validation.

Trusted-source layer in the experience

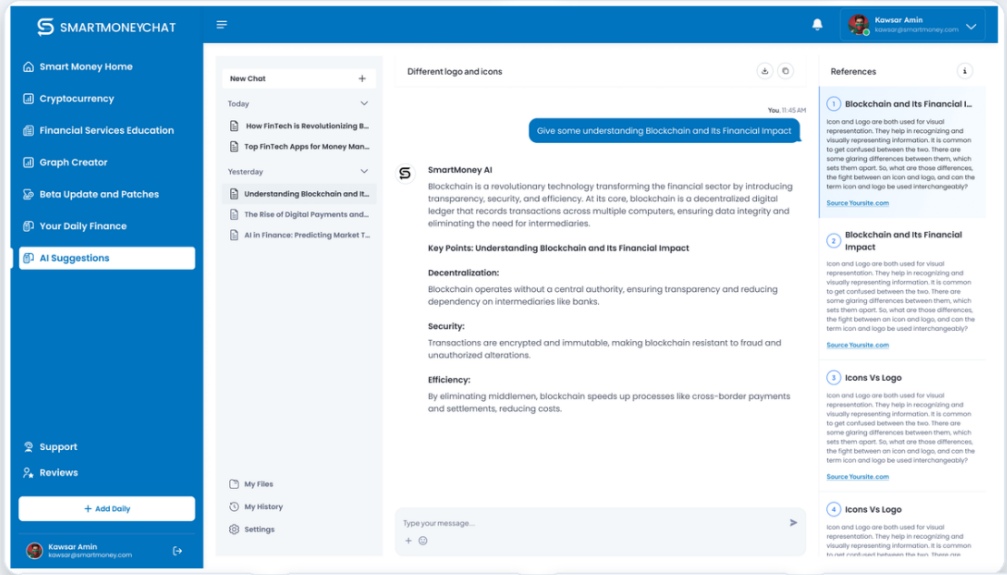
In Smart Money Chat, I framed the experience around trusted financial sources such as Bloomberg, CNBC, The Wall Street Journal, Yahoo Finance, and live market feeds.

Primary interaction path

I used the prompt flow as a key test path for send actions, keyboard controls, voice-text behaviour, and API error states.

Authenticated workspace

I validated profile context and session state as part of identity, access, and authorization checks.



Modular finance workspace

I designed and tested multiple entry points here: home, crypto, education, graph creator, daily finance, and AI suggestions.

Session and chat history

I treated saved threads and continuity as part of the QA surface across navigation, reload behaviour, and state persistence.

AI response rendering

I viewed this response area as the core value surface, with focus on clarity, formatting, loading states, and fallback behaviour.

Source-backed trust layer

I used references to reinforce trust and finance education, while validating source labels, ordering, and rendering stability.

How this reflects my work

In this project, I worked across front-end delivery and QA-facing validation using React, TypeScript, NestJS, Azure OpenAI, LangChain with RAG, and secure market-data requests. This screen also shows the testing depth I handled around UI behaviour, API responses, authentication state, source rendering, and regression coverage across a finance-focused SaaS platform.

What this screen shows

In Smart Money Chat, I brought together authenticated AI chat, finance workflows, source-backed answers, and market-data retrieval in one interface. That gives this screen strong value from both a product and QA perspective.

It also reflects a RAG-style backend flow: user prompt, retrieval from trusted finance content and API feeds, structured answer generation, and references rendered in the UI.

Representative NestJS + Azure OpenAI service

```
@Injectable()
export class FinanceChatService {
constructor(
private readonly vectorStore: FinanceVectorStore,
private readonly marketDataService: MarketDataService,
private readonly azureOpenAI: AzureChatOpenAI,
) {}
async answerQuestion(question: string) {
const docs = await this.vectorStore.similaritySearch(question, 4);
const market = await this.marketDataService.lookupTickerContext(question);
```

Reflects my work with NestJS, Azure OpenAI, retrieval logic, and API integration.

Representative LangChain prompt / retrieval flow

```
const prompt = ChatPromptTemplate.fromTemplate(`
You are Smart Money Chat, an educational finance assistant.
Use retrieved context and live market data only.
Explain clearly. Return references users can verify.
Question: {question}
Context: {context}
Market Data: {market}
`);
const chain = RunnableSequence.from([
```

Shows the kind of controlled finance prompting and source-aware output I worked around.

Live market-data lookup context

```
async lookupTickerContext(question: string) {
const symbol = extractSymbolOrName(question);
const quote = await axios.get(`/quotes?symbol=${symbol}`);
const exchanges = ['NASDAQ', 'LSE', 'NYSE'];
return {
```

Links to real-time stock and ETF queries across NASDAQ, LSE, and NYSE feeds.

Smart Money Chat | Cryptocurrency Dashboard, Live Market Data, and QA-Ready Analytics UI

This page highlights my work across live crypto market feeds, typed React dashboard state, API-driven analytics UI, news integration, and QA validation of data-rich financial surfaces.

What this screen demonstrates

In Smart Money Chat, I worked around a cryptocurrency dashboard that combines real-time market summaries, sparkline-based trend cards, a large tabular data surface, pagination, category filtering, and a crypto newsfeed in one consistent interface. From a QA perspective, this screen is valuable because it concentrates live values, rendering logic, component reuse, and state-heavy interactions in one place.

QA surfaces I focused on

I treated this page as a strong validation surface for price formatting, card consistency, filter state, page transitions, row stability, graph rendering, image handling, and responsive layout behaviour across dense financial content.

How it matches my portfolio

This screen reflects the kind of work I want to show: React and TypeScript front-end delivery, API-driven financial interfaces, dashboard QA, live-data rendering, and disciplined testing across reusable SaaS components.

Representative market-data fetch and mapping

```ts
export async function fetchCryptoMarkets(params: MarketParams) {
const response = await axios.get('/api/crypto/markets', { params });
return response.data.map((coin) => ({
symbol: coin.symbol,
name: coin.name,
price: coin.currentPrice,
marketCap: coin.marketCap,
```

Shows API-driven market data handling and typed front-end mapping.

Representative React state and pagination flow

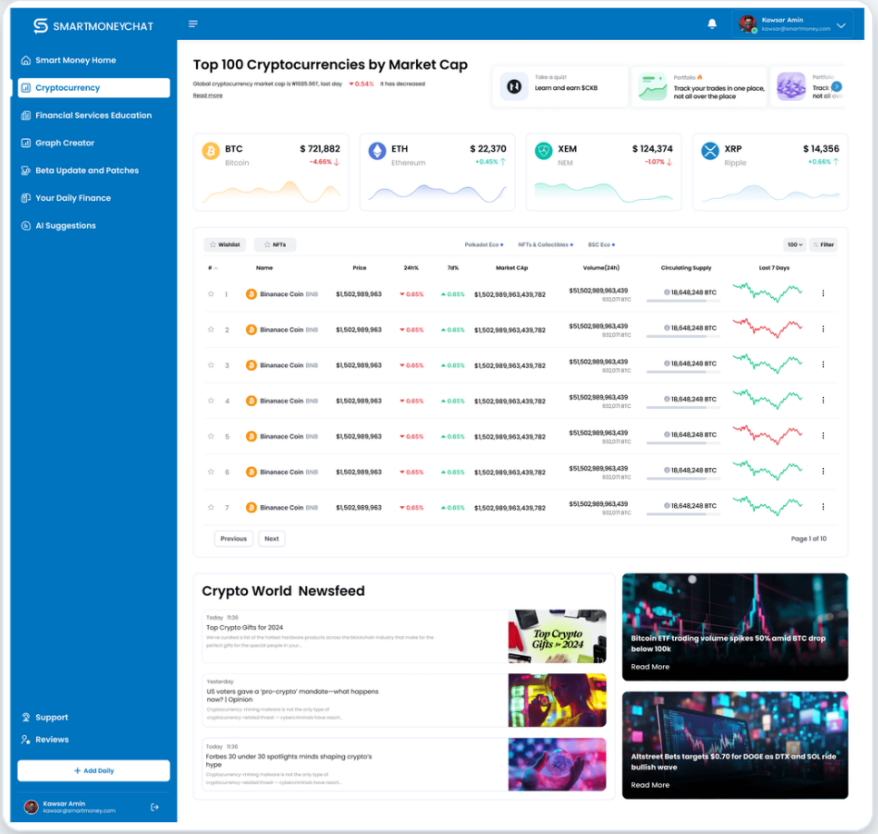
```ts
const visibleRows = useMemo(() => {
return markets
.filter((row) => activeTab === 'all' || row.category === activeTab)
.slice((page - 1) * pageSize, page * pageSize);
}, [markets, activeTab, page, pageSize]);
return visibleRows.map((coin) => (
<CryptoRow
```

Shows the state logic behind filtering, views, and paginated table rendering.

Representative crypto newsfeed integration

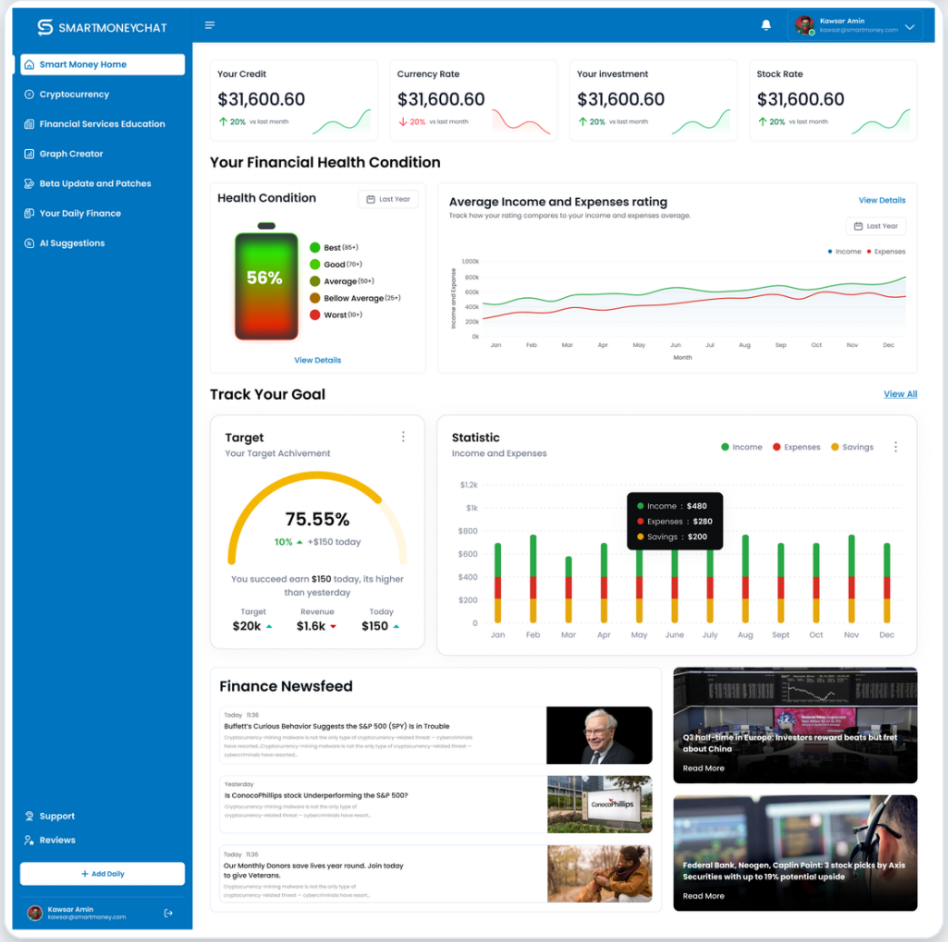
```ts
async function fetchCryptoNews() {
const { data } = await axios.get('/api/news/crypto');
return data.articles.map((article) => ({
title: article.title,
```

Connects the dashboard to news-card rendering and QA validation of media content.



Smart Money Chat | Financial Health Dashboard, FastAPI Market Stream, and Trusted News Ingestion

This page highlights my work across Python FastAPI services, live market and trusted-news ingestion, dashboard payload assembly, and QA validation of finance-focused front-end components.



What this screen demonstrates

In Smart Money Chat, I worked around a financial health dashboard that combines KPI cards, comparative charting, goal tracking, savings statistics, and a finance newsfeed inside one product surface. For this slide, I wanted to emphasise the Python and FastAPI side that can stream live market data and trusted financial news into these front-end components.

Trusted-source data design

I tied the news and market layer to high-credibility sources such as CNBC, Bloomberg, Yahoo Finance, and Morningstar so the dashboard can present timely and trusted financial context rather than generic feed content.

QA depth in this view

I treated live values, chart consistency, card states, tooltips, image handling, and content refresh timing as core validation points across this dashboard.

Representative Python FastAPI market stream

```python
from fastapi import APIRouter
import httpx
router = APIRouter(prefix='/market')
@router.get('/stream')
async def stream_market_snapshot(symbol: str):
async with httpx.AsyncClient(timeout=10.0) as client:
quote = await client.get(f'https://api.example.com/quote/{symbol}')
```

Shows async FastAPI-style aggregation of live quote and news payloads.

Representative trusted-source filtering

```python
TRUSTED_NEWS_SOURCES = {
'CNBC',
'Bloomberg',
'Yahoo Finance',
'Morningstar',
}
```

Keeps dashboard news aligned to CNBC, Bloomberg, Yahoo Finance, and Morningstar.

Representative dashboard payload assembly

```python
async def build_financial_dashboard(symbol: str):
snapshot = await stream_market_snapshot(symbol)
articles = filter_trusted_articles(snapshot['news']['articles'])
return {
'summaryCards': derive_summary_cards(snapshot['quote']),
```

Connects Python service output to front-end cards, charts, goals, and newsfeed rendering.

Smart Money Chat | Daily Finance System, TypeScript UI State, and Python / Azure Data Services

This page highlights my work across TypeScript CRUD state, Python FastAPI validation, Cosmos DB persistence, Redis cache freshness, Blob audit storage, and QA coverage for finance workflows.

What this screen demonstrates

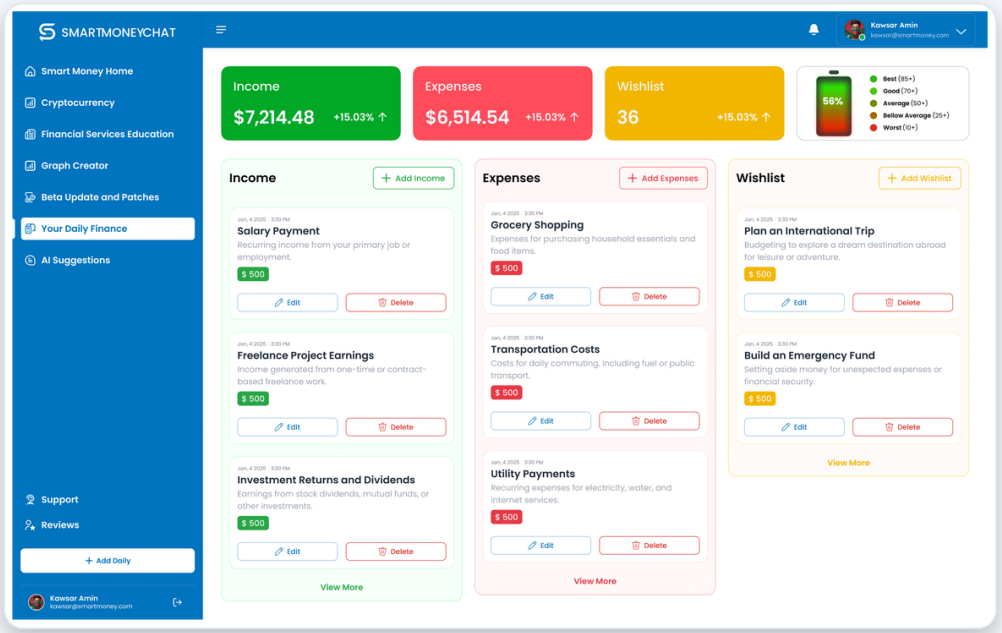
In Smart Money Chat, I designed and validated a Daily Finance workflow that lets users manage income, expenses, and wishlists inside one system. The engineering value is in the chain behind the interface: typed React components, reliable CRUD actions, deterministic backend validation, cache-aware refresh behaviour, and persistent storage for user financial records.

TypeScript and UI state

I used modular React cards, typed form handling, optimistic updates, and reusable CRUD state to keep the workflow fast and production-ready.

Backend and QA depth

I treated edit, delete, save, stale-data, and refresh timing as high-value validation points across FastAPI, Redis, Cosmos DB, and Blob-backed flows.



Representative TypeScript Redux CRUD flow

```ts
export const saveDailyFinanceEntry = createAsyncThunk(
'dailyFinance/saveEntry',
async (payload: DailyFinancePayload) => {
const { data } = await api.post('/daily-finance/entries', payload);
return data as DailyFinanceEntry;
}
);
builder.addCase(saveDailyFinanceEntry.fulfilled, (state, action) => {
upsertEntry(state.entries, action.payload);
```

Shows typed async state handling, selectors, and reusable card-level updates.

Representative Python FastAPI validation layer

```python
from fastapi import APIRouter
from pydantic import BaseModel, Field
class FinanceEntry(BaseModel):
user_id: str
category: str
title: str = Field(min_length=2, max_length=120)
amount: float = Field(gt=0)
@router.post('/daily-finance/entries')
async def create_entry(entry: FinanceEntry):
```

Shows Python validation before persisting financial entries into backend workflows.

Representative Cosmos DB + Redis + Blob service

```python
async def create_finance_entry(entry: dict):
cache_key = f"daily-finance:{entry['user_id']}"
audit_name = f"audit/{entry['user_id']}/{entry['id']}.json"
await cosmos_container.upsert_item(entry)
await redis_client.delete(cache_key)
await blob_client.upload_blob(
name=audit_name,
data=json.dumps(entry),
overwrite=True
```

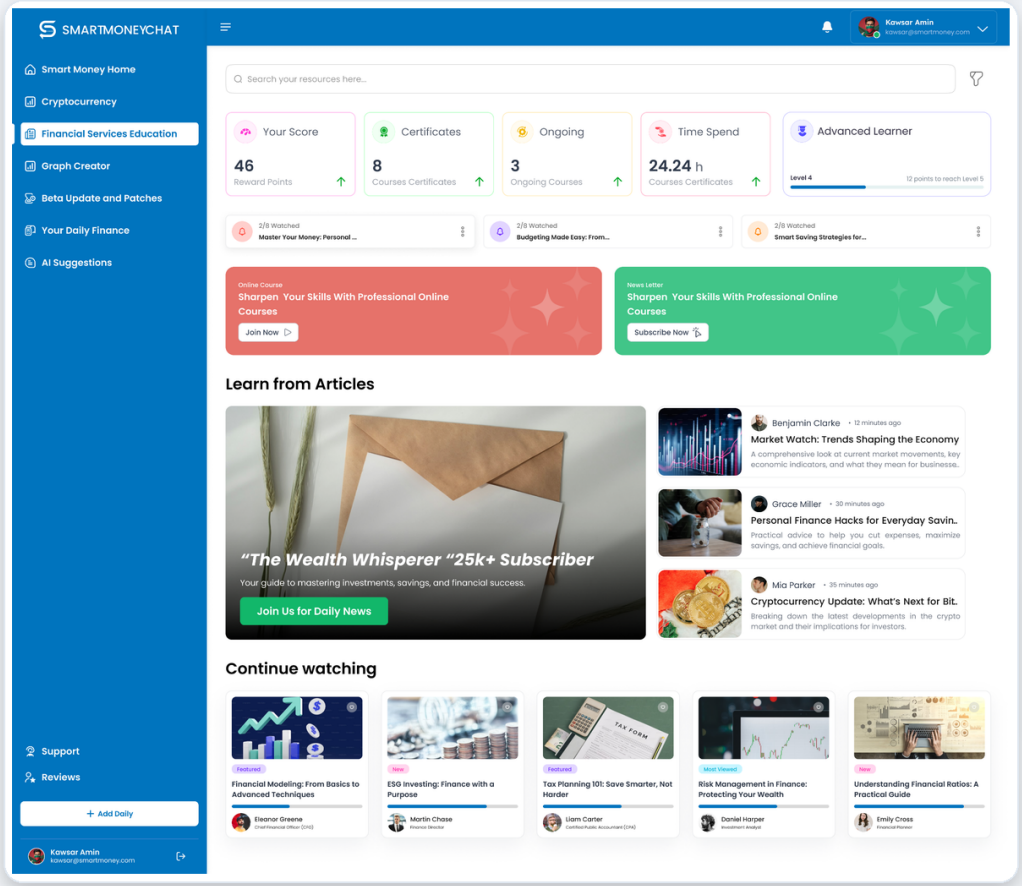
Shows persistence, cache freshness, and audit-storage traceability recruiters look for.

How this reflects my work

This page brings together the front-end and platform strengths on my CV: React, TypeScript, FastAPI, REST APIs, Redis, Cosmos DB, Blob Storage, and QA-oriented thinking around edit flows, delete safety, stale data, and reliable user state.

Smart Money Chat | Study Room, Personalized Learning, and Azure-Backed Education Services

This page highlights my work across TypeScript search and progress state, Python recommendation services, Redis-backed personalization, Cosmos DB learning persistence, Blob-delivered content assets, and QA validation for education-first finte



What this screen demonstrates

In Smart Money Chat, I used the Study Room to connect fintech with financial education through guided content, article discovery, learner scoring, certificate progress, continue-watching flows, and personalized recommendations. This page naturally extends the earlier technical story by adding search quality, recommendation logic, user progress persistence, and content delivery services on top of the platform foundations already shown.

New strengths highlighted here

I use this slide to show typed search flows, resource ranking, learning analytics, recommendation scoring, resume-state accuracy, and user-focused content orchestration.

Portfolio relevance

This page strengthens the overall case study by showing that my work also covers education-first product systems where personalization, progress tracking, and reliable content delivery matter as much as the interface itself.

Representative TypeScript search and progress flow

```ts
const debouncedQuery = useDebounce(searchTerm, 250);
const visibleResources = useMemo(() => {
return resources
.filter((item) => matchesQuery(item, debouncedQuery))
.map((item) => ({
...item,
progressPct: progressById[item.id] ?? 0,
score: rankForUser(item, learnerProfile)
```

Shows typed search, recommendation scoring, and continue-watching state updates.

Representative Python recommendation endpoint

```python
from fastapi import APIRouter
from pydantic import BaseModel
class RecommendationRequest(BaseModel):
user_id: str
query: str | None = None
interests: list[str] = []
level: int = 1
```

Shows a FastAPI-style service that ranks education resources for the user.

Representative Cosmos DB + Redis + Blob persistence

```python
async def persist_learning_state(user_id: str, resource_id: str, progress: int):
cache_key = f'study-room:{user_id}:dashboard'
await cosmos_container.upsert_item({
'id': f'{user_id}:{resource_id}',
'userId': user_id,
```

Shows Azure-backed progress persistence, cache freshness, and asset metadata.

Smart Money Chat | Portfolio Summary and Closing Overview

A unified view of the AI, fintech, data-platform, and education systems I designed and validated across Smart Money Chat.

Closing summary

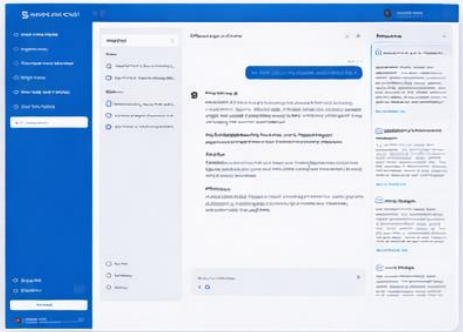
Across Smart Money Chat, I worked on AI-assisted financial education, live crypto and market dashboards, financial health systems, and personalized Study Room experiences. Together, these pages show how combine React and TypeScript frontend engineering with modern FastAPI services, Azure-backed data infrastructure, and QA-focused validation across finance-critical user journeys.

Core technologies demonstrated

React	TypISPI	Redis	Python
FastAPI	Next.js	Python	RESTIul servers
REST APIs	QA Validation	Azure Services	Scalable SaaS

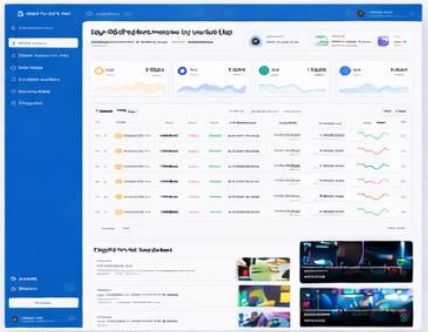
01 AI Financial Research

TypeScript React, AI chat trusted-source research software, QA-ready product validation



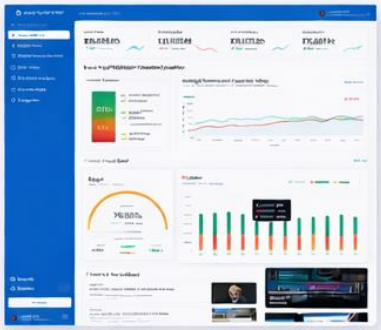
02 Live Crypto Dashboard

TypeScript React, live market data, dashboard interfaces, trade feeds, analytics



03 Financial Health Dashboard

FastAPI market data, trusted news, validation



04 Study Room Personalization

TypeScript recommendation state, scoring, progress presentation, education-first product systems

05 Daily Finance System

TypeScript CRUD-UI, daily finance, account workflows, health automation, Cosmos DB, Redis, BIOB cloud flow

Thank you

Thank you for reviewing my **Smart Money Chat** portfolio.

This body of work reflects my approach to building secure, scalable, and user-focused fintech systems that comaine strong engineering foundations with clear product thinking and disciplined QA validation.