### Project Report: Refactor Royale AI-Code Optimization Game

**Title:** Refactor Royale AI-Code Optimization Game

**Team Members:**

* Vinutha S - 1SB22IS060
* Kokila M - 1SB22IS020

# Abstract

Refactor Royale is an AI-driven code optimization game designed to enhance programming skills through interactive challenges. Players refine inefficient code using AI-generated suggestions, focusing on performance, readability, and best practices. The game employs machine learning to analyze and improve code iteratively, offering real-time feedback and ranking systems. With an engaging, competitive format, Refactor Royale transforms code optimization into a dynamic, educational experience for developers of all levels.

# Problem Statement

In software development, writing optimal, clean, and efficient code is a crucial skill. However, many developers struggle with understanding best practices for refactoring, performance optimization, and reducing technical debt. Traditional learning methods, such as books and tutorials, lack engagement and real-time feedback, making it difficult for developers to improve their code refactoring skills interactively.

# Solution

Refactor Royale is an AI-driven competitive coding game where players are given pre- written code with inefficiencies and must refactor it to maximize efficiency, readability, and maintainability. The system evaluates solutions using AI-based analysis, scoring based on execution time, space complexity, and coding best practices.

# Code Base

## Technology Stack

* **Frontend:** React.js, Vite, CodeMirror, React Confetti, CSS3
* **Backend:** Flask, Flask-CORS, Python AST, Subprocess
* **AI & Optimization Engine:** Ollama (Mistral 7B), Tenacity, LRU Cache
* **Database:** SQL (Users Table, Challenges Table, Submissions Table)

**CODE**

**Frontend : app.css**

/\* Base Styles \*/

:root {

  --neon-cyan: #0ff0fc;

  --neon-magenta: #ff00ff;

  --neon-purple: #9d00ff;

  --neon-blue: #00b4ff;

  --neon-green: #00ff9d;

  --neon-red: #ff2d75;

  --neon-yellow: #fff000;

  --dark-bg: #0a0a12;

  --darker-bg: #050508;

  --glass-bg: rgba(15, 15, 25, 0.5);

}

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

  font-family: 'Courier New', 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

}

body {

  background-color: var(--dark-bg);

  color: white;

  overflow-x: hidden;

}

/\* App Container \*/

.app-container {

  min-height: 100vh;

  display: flex;

  flex-direction: column;

  position: relative;

  overflow: hidden;

}

.app-container.login-glow {

  animation: loginGlow 1s ease-out;

}

@keyframes loginGlow {

  0% { background-color: var(--dark-bg); }

  50% { background-color: rgba(0, 180, 255, 0.1); }

  100% { background-color: var(--dark-bg); }

}

/\* Cyber Grid Background \*/

.cyber-grid {

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  display: grid;

  grid-template-columns: repeat(5, 1fr);

  grid-template-rows: repeat(4, 1fr);

  z-index: -1;

  opacity: 0.1;

}

.grid-line {

  border-right: 1px solid rgba(0, 255, 252, 0.1);

  border-bottom: 1px solid rgba(0, 255, 252, 0.1);

}

/\* Glassmorphic Elements \*/

.glassmorphic {

  background: var(--glass-bg);

  backdrop-filter: blur(10px);

  -webkit-backdrop-filter: blur(10px);

  border-radius: 10px;

  border: 1px solid rgba(255, 255, 255, 0.1);

  box-shadow: 0 8px 32px rgba(0, 0, 0, 0.2);

}

/\* Login Page \*/

.login-wrapper {

  display: flex;

  justify-content: center;

  align-items: center;

  min-height: 100vh;

  padding: 20px;

}

.login-container {

  width: 100%;

  max-width: 500px;

  padding: 40px 30px;

  text-align: center;

  position: relative;

  overflow: hidden;

  animation: fadeIn 0.5s ease-out;

  border: 2px solid var(--neon-blue);

  box-shadow: 0 0 15px var(--neon-blue),

              inset 0 0 10px var(--neon-blue);

}

.login-container::before {

  content: '';

  position: absolute;

  top: -10px;

  left: -10px;

  right: -10px;

  bottom: -10px;

  border: 2px solid var(--neon-blue);

  border-radius: 12px;

  animation: pulseGlow 2s infinite alternate;

  pointer-events: none;

  z-index: -1;

}

@keyframes fadeIn {

  from { opacity: 0; transform: translateY(20px); }

  to { opacity: 1; transform: translateY(0); }

}

.neon-title {

  font-size: 2.5rem;

  margin-bottom: 30px;

  text-transform: uppercase;

  letter-spacing: 3px;

  color: var(--neon-cyan);

  text-shadow: 0 0 10px var(--neon-cyan),

               0 0 20px var(--neon-cyan),

               0 0 40px var(--neon-cyan);

  animation: neonPulse 2s infinite alternate;

}

@keyframes neonPulse {

  from { text-shadow: 0 0 5px var(--neon-cyan),

                     0 0 10px var(--neon-cyan); }

  to { text-shadow: 0 0 10px var(--neon-cyan),

                   0 0 20px var(--neon-cyan),

                   0 0 40px var(--neon-cyan); }

}

.input-container {

  position: relative;

  margin: 25px 0;

}

.neon-input {

  width: 100%;

  padding: 15px 20px;

  background: rgba(10, 10, 20, 0.7);

  border: 1px solid var(--neon-cyan);

  border-radius: 5px;

  color: white;

  font-size: 1rem;

  outline: none;

  transition: all 0.3s ease;

}

.neon-input:focus {

  box-shadow: 0 0 10px var(--neon-cyan),

              inset 0 0 5px var(--neon-cyan);

}

.input-highlight {

  position: absolute;

  bottom: 0;

  left: 0;

  width: 0;

  height: 2px;

  background: var(--neon-cyan);

  transition: all 0.3s ease;

}

.neon-input:focus + .input-highlight {

  width: 100%;

}

/\* Buttons \*/

.neon-button {

  position: relative;

  padding: 12px 25px;

  margin: 10px 5px;

  background: transparent;

  color: white;

  border: 2px solid;

  border-radius: 5px;

  font-size: 1rem;

  font-weight: bold;

  text-transform: uppercase;

  letter-spacing: 1px;

  cursor: pointer;

  overflow: hidden;

  transition: all 0.3s ease;

  outline: none;

}

.neon-button .button-text {

  position: relative;

  z-index: 2;

}

.neon-button .button-glow {

  position: absolute;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  background: linear-gradient(90deg, transparent, rgba(255,255,255,0.2), transparent);

  transform: translateX(-100%);

  transition: all 0.6s ease;

}

.neon-button:hover .button-glow {

  transform: translateX(100%);

}

.neon-button-cyan {

  border-color: var(--neon-cyan);

  color: var(--neon-cyan);

  box-shadow: 0 0 10px rgba(15, 240, 252, 0.5),

              inset 0 0 5px rgba(15, 240, 252, 0.5);

}

.neon-button-magenta {

  border-color: var(--neon-magenta);

  color: var(--neon-magenta);

  box-shadow: 0 0 10px rgba(255, 0, 255, 0.5),

              inset 0 0 5px rgba(255, 0, 255, 0.5);

}

.neon-button-blue {

  border-color: var(--neon-blue);

  color: var(--neon-blue);

  box-shadow: 0 0 10px rgba(0, 180, 255, 0.5),

              inset 0 0 5px rgba(0, 180, 255, 0.5);

}

.neon-button-purple {

  border-color: var(--neon-purple);

  color: var(--neon-purple);

  box-shadow: 0 0 10px rgba(157, 0, 255, 0.5),

              inset 0 0 5px rgba(157, 0, 255, 0.5);

}

.neon-button:hover {

  transform: translateY(-2px);

  box-shadow: 0 5px 15px rgba(0, 180, 255, 0.4);

}

.neon-button:active {

  transform: translateY(1px);

}

/\* Game UI \*/

.game-wrapper {

  padding: 20px;

  max-width: 1200px;

  margin: 0 auto;

  width: 100%;

}

.game-container {

  padding: 30px;

  margin-bottom: 30px;

}

.game-header {

  margin-bottom: 30px;

  text-align: center;

}

.stats-container {

  display: flex;

  justify-content: space-around;

  flex-wrap: wrap;

  margin: 20px 0;

  gap: 15px;

}

.stat-box {

  padding: 10px 15px;

  min-width: 120px;

  text-align: center;

}

.stat-label {

  display: block;

  font-size: 0.8rem;

  color: rgba(255, 255, 255, 0.7);

  margin-bottom: 5px;

}

.stat-value {

  font-size: 1.2rem;

  font-weight: bold;

}

.neon-cyan { color: var(--neon-cyan); }

.neon-purple { color: var(--neon-purple); }

.neon-green { color: var(--neon-green); }

.neon-red { color: var(--neon-red); }

.flicker {

  animation: flicker 1.5s infinite alternate;

}

@keyframes flicker {

  0%, 19%, 21%, 23%, 25%, 54%, 56%, 100% {

    text-shadow: 0 0 5px currentColor,

                 0 0 10px currentColor;

  }

  20%, 24%, 55% {

    text-shadow: none;

  }

}

/\* Challenge Card \*/

.challenge-card {

  padding: 20px;

  margin-bottom: 20px;

}

.card-header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  margin-bottom: 15px;

}

.difficulty-meter {

  display: flex;

  gap: 5px;

}

.difficulty-meter span {

  display: inline-block;

  width: 10px;

  height: 10px;

  border-radius: 50%;

  background: rgba(255, 255, 255, 0.1);

}

.difficulty-meter span.active {

  background: var(--neon-yellow);

  box-shadow: 0 0 5px var(--neon-yellow);

}

.neon-code {

  font-family: 'Courier New', monospace;

  white-space: pre-wrap;

  line-height: 1.5;

  color: rgba(255, 255, 255, 0.9);

  text-shadow: 0 0 2px rgba(255, 255, 255, 0.5);

}

/\* Editor \*/

.editor-container {

  position: relative;

  margin-bottom: 20px;

}

.editor-corner {

  position: absolute;

  width: 20px;

  height: 20px;

  border-style: solid;

  border-color: var(--neon-cyan);

  border-width: 0;

}

.editor-corner:nth-child(1) {

  top: 0;

  left: 0;

  border-top-width: 2px;

  border-left-width: 2px;

}

.editor-corner:nth-child(2) {

  top: 0;

  right: 0;

  border-top-width: 2px;

  border-right-width: 2px;

}

.editor-corner:nth-child(3) {

  bottom: 0;

  right: 0;

  border-bottom-width: 2px;

  border-right-width: 2px;

}

.editor-corner:nth-child(4) {

  bottom: 0;

  left: 0;

  border-bottom-width: 2px;

  border-left-width: 2px;

}

/\* Button Group \*/

.button-group {

  display: flex;

  justify-content: center;

  gap: 15px;

  margin: 25px 0;

}

/\* Feedback Card \*/

.feedback-card {

  padding: 20px;

  margin: 20px 0;

}

.feedback-text {

  margin: 15px 0;

  line-height: 1.6;

}

.score-badge {

  display: inline-block;

  padding: 5px 10px;

  border-radius: 20px;

  font-weight: bold;

  font-size: 1.2rem;

  animation: pulse 1.5s infinite;

}

@keyframes pulse {

  0% { transform: scale(1); }

  50% { transform: scale(1.05); }

  100% { transform: scale(1); }

}

/\* Solution Card \*/

.solution-card {

  padding: 20px;

  margin: 20px 0;

}

/\* Loading Spinner \*/

.loading-container {

  text-align: center;

  margin: 20px 0;

}

.neon-spinner {

  width: 50px;

  height: 50px;

  border-radius: 50%;

  position: relative;

  margin: 0 auto 15px;

  animation: spin 1.5s linear infinite;

}

.spinner-sector {

  position: absolute;

  width: 100%;

  height: 100%;

  border-radius: 50%;

  border: 5px solid transparent;

  mix-blend-mode: overlay;

}

.spinner-sector-cyan {

  border-top-color: var(--neon-cyan);

  --rotation: 0;

  animation: spinReverse 1s linear infinite;

}

.spinner-sector-magenta {

  border-left-color: var(--neon-magenta);

  --rotation: 120;

  animation: spinReverse 1.5s linear infinite;

}

.spinner-sector-blue {

  border-right-color: var(--neon-blue);

  --rotation: 240;

  animation: spinReverse 2s linear infinite;

}

@keyframes spin {

  to { transform: rotate(360deg); }

}

@keyframes spinReverse {

  to { transform: rotate(-360deg); }

}

/\* Game Over Screen \*/

.game-over {

  padding: 40px;

  text-align: center;

}

.congratulations {

  margin-bottom: 30px;

}

.neon-flicker {

  font-size: 2.5rem;

  margin-bottom: 30px;

  color: var(--neon-magenta);

  text-shadow: 0 0 10px var(--neon-magenta);

  animation: neonFlicker 1.5s infinite alternate;

}

@keyframes neonFlicker {

  0%, 19%, 21%, 23%, 25%, 54%, 56%, 100% {

    text-shadow: 0 0 5px var(--neon-magenta),

                 0 0 10px var(--neon-magenta),

                 0 0 20px var(--neon-magenta);

  }

  20%, 24%, 55% {

    text-shadow: none;

  }

}

.final-stats {

  max-width: 400px;

  margin: 0 auto;

}

.stat-row {

  display: flex;

  justify-content: space-between;

  padding: 10px 0;

  border-bottom: 1px solid rgba(255, 255, 255, 0.1);

}

.restart-buttons {

  margin-top: 30px;

}

/\* Error Messages \*/

.error {

  color: var(--neon-red);

  margin: 15px 0;

  text-align: center;

  text-shadow: 0 0 5px rgba(255, 45, 117, 0.5);

}

/\* Login Footer \*/

.login-footer {

  margin-top: 30px;

}

.neon-text {

  color: rgba(255, 255, 255, 0.7);

  font-size: 0.9rem;

  margin: 5px 0;

  text-transform: uppercase;

  letter-spacing: 1px;

}

/\* Responsive Adjustments \*/

@media (max-width: 768px) {

  .game-container, .login-container {

    padding: 20px 15px;

  }

  .stats-container {

    flex-direction: column;

    align-items: center;

  }

  .button-group {

    flex-direction: column;

  }

  .neon-title {

    font-size: 1.8rem;

  }

}

**Frontend : app.js**

import { useState, useEffect } from "react";

import Confetti from "react-confetti";

import CodeMirror from "@uiw/react-codemirror";

import { python } from "@codemirror/lang-python";

import "./App.css";

function App() {

const [username, setUsername] = useState("");

const [isPlaying, setIsPlaying] = useState(false);

const [code, setCode] = useState("");

const [feedback, setFeedback] = useState(null);

const [score, setScore] = useState(null);

const [solution, setSolution] = useState(null);

const [loading, setLoading] = useState(false);

const [error, setError] = useState(null);

const [challenge, setChallenge] = useState("");

const [challengeId, setChallengeId] = useState(null);

const [solvedCount, setSolvedCount] = useState(0);

const [totalScore, setTotalScore] = useState(0);

const [gameTimeLeft, setGameTimeLeft] = useState(900);

const [isGameOver, setIsGameOver] = useState(false);

const [showConfetti, setShowConfetti] = useState(false);

const [showLoginGlow, setShowLoginGlow] = useState(false);

const [sessionId, setSessionId] = useState(null);

const [showScores, setShowScores] = useState(false);

const [questionScores, setQuestionScores] = useState([]);

const API\_BASE\_URL = "http://localhost:5001";

const startGame = async () => {

if (!username.trim()) {

setError("Please enter a username!");

return;

}

setShowLoginGlow(true);

try {

const response = await fetch(`${API\_BASE\_URL}/register`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({ username }),

});

if (!response.ok) {

const errorData = await response.json();

throw new Error(errorData.error || "Registration failed");

}

const data = await response.json();

setSessionId(data.session\_id);

setTimeout(() => {

setIsPlaying(true);

fetchChallenge(true);

setShowLoginGlow(false);

}, 1000);

} catch (err) {

setError(err.message);

}

};

const fetchChallenge = async (isFirst = false, retryCount = 0) => {

if (isGameOver) return;

setLoading(true);

setError(null);

try {

const response = await fetch(`${API\_BASE\_URL}/get\_challenge`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

username,

session\_id: sessionId

}),

});

if (response.status === 401) {

setError("Session expired. Please refresh.");

setIsPlaying(false);

return;

}

if (response.status === 404) {

throw new Error("No challenges available at this time");

}

if (!response.ok) {

throw new Error(`Server responded with ${response.status}`);

}

const data = await response.json();

if (data.error) {

throw new Error(data.error);

}

if (data.completed && data.progress?.answered >= 10) {

endGame(true);

return;

}

setChallenge(data.challenge.code);

setChallengeId(data.challenge.id);

setFeedback(null);

setScore(null);

setSolution(null);

setCode("");

if (!isFirst) {

setSolvedCount(prev => {

const newCount = data.progress?.answered || prev + 1;

return newCount;

});

setGameTimeLeft(Math.floor(data.time\_remaining || gameTimeLeft));

}

} catch (err) {

if (retryCount < 2) {

setTimeout(() => fetchChallenge(isFirst, retryCount + 1), 1000);

} else {

setError(`Failed to load challenge: ${err.message}`);

if (err.message.includes("No challenges available")) {

endGame(false);

}

}

} finally {

setLoading(false);

}

};

const evaluateCode = async () => {

if (!code.trim()) {

setError("Please enter code to evaluate!");

return;

}

if (!code.includes("def ") || !code.includes("(") || !code.includes(")")) {

setError("Code must contain a function definition");

return;

}

setLoading(true);

setError(null);

try {

const response = await fetch(`${API\_BASE\_URL}/evaluate`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

username,

session\_id: sessionId,

code,

challenge\_code: challenge,

challenge\_id: challengeId,

}),

});

if (response.status === 401) {

setError("Session expired. Please refresh.");

setIsPlaying(false);

return;

}

if (!response.ok) {

const errorData = await response.json();

throw new Error(errorData.error || "Evaluation failed");

}

const data = await response.json();

setFeedback(data.feedback);

setScore(data.score);

setTotalScore(data.total\_score);

} catch (err) {

setError(err.message);

} finally {

setLoading(false);

}

};

const fetchSolution = async () => {

setLoading(true);

setError(null);

try {

const response = await fetch(`${API\_BASE\_URL}/get\_solution`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

username,

session\_id: sessionId,

code: challenge,

}),

});

if (response.status === 401) {

setError("Session expired. Please refresh.");

setIsPlaying(false);

return;

}

if (!response.ok) {

const errorData = await response.json();

throw new Error(errorData.error || "Failed to fetch solution");

}

const data = await response.json();

setSolution(data.solution || challenge);

} catch (err) {

setError(err.message);

setSolution(challenge);

} finally {

setLoading(false);

}

};

const fetchQuestionScores = async () => {

try {

const response = await fetch(`${API\_BASE\_URL}/get\_all\_questions`, {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

username,

session\_id: sessionId

}),

});

if (!response.ok) {

throw new Error("Failed to fetch question scores");

}

const data = await response.json();

if (data.questions) {

const scores = data.questions

.filter(q => q.attempted)

.sort((a, b) => a.id - b.id)

.map((q, index) => ({

displayNumber: index + 1,

originalId: q.id,

score: q.score

}));

setQuestionScores(scores);

}

} catch (err) {

setError("Failed to load question scores");

console.error(err);

}

};

const handleNextOrFinish = async () => {

if (solvedCount >= 9) {

endGame(true);

return;

}

if (!feedback) {

setError("Please evaluate code first!");

return;

}

await fetchChallenge();

};

const endGame = (isComplete = false) => {

setIsGameOver(true);

if (isComplete && solvedCount >= 9) {

setShowConfetti(true);

setSolvedCount(10);

}

};

useEffect(() => {

if (!isPlaying || isGameOver) return;

const interval = setInterval(() => {

setGameTimeLeft((prev) => {

if (prev <= 1) {

clearInterval(interval);

endGame(false);

return 0;

}

return prev - 1;

});

}, 1000);

return () => clearInterval(interval);

}, [isPlaying, isGameOver]);

const formatTime = (seconds) => {

const minutes = Math.floor(seconds / 60);

const secs = seconds % 60;

return `${minutes}:${secs.toString().padStart(2, "0")}`;

};

return (

<div className={`app-container ${showLoginGlow ? 'login-glow' : ''}`}>

{showConfetti && <Confetti recycle={false} numberOfPieces={500} />}

{!isPlaying ? (

<div className="login-wrapper">

<div className="login-container glassmorphic">

<h1 className="neon-title">REFACTOR ROYALE</h1>

<div className="input-container">

<input

type="text"

placeholder="ENTER USERNAME"

value={username}

onChange={(e) => setUsername(e.target.value)}

className="neon-input"

onKeyPress={(e) => e.key === 'Enter' && startGame()}

/>

<span className="input-highlight"></span>

</div>

<button

onClick={startGame}

className="neon-button neon-button-cyan"

>

<span className="button-text">START CODING</span>

<span className="button-glow"></span>

</button>

{error && <p className="error">{error}</p>}

</div>

</div>

) : (

<div className="game-wrapper">

<div className="game-container glassmorphic">

<div className="game-header">

<h1 className="neon-title">CODE OPTIMIZER</h1>

<div className="stats-container">

<div className="stat-box">

<span className="stat-label">PLAYER</span>

<span className="stat-value neon-cyan">{username}</span>

</div>

<div className="stat-box">

<span className="stat-label">SOLVED</span>

<span className="stat-value neon-purple">{solvedCount}/10</span>

</div>

<div className="stat-box">

<span className="stat-label">SCORE</span>

<span className="stat-value neon-green">{totalScore}</span>

</div>

<div className="stat-box">

<span className="stat-label">TIME</span>

<span className="stat-value neon-red">{formatTime(gameTimeLeft)}</span>

</div>

</div>

</div>

{!isGameOver ? (

<>

<div className="challenge-card glassmorphic">

<div className="card-header">

<h3>CHALLENGE</h3>

<div className="difficulty-meter">

{Array.from({ length: 5 }).map((\_, i) => (

<span key={i} className={i < (solvedCount % 5) ? 'active' : ''}></span>

))}

</div>

</div>

<pre className="neon-code">{challenge || "LOADING CHALLENGE..."}</pre>

</div>

<div className="editor-container">

<CodeMirror

value={code}

height="200px"

extensions={[python()]}

onChange={(value) => setCode(value)}

placeholder="// OPTIMIZE THIS CODE..."

theme="dark"

/>

</div>

<div className="button-group">

<button

onClick={evaluateCode}

disabled={loading}

className="neon-button neon-button-blue"

>

OPTIMIZE

</button>

<button

onClick={handleNextOrFinish}

disabled={!feedback || loading}

className="neon-button neon-button-magenta"

>

{solvedCount >= 9 ? "FINISH" : "NEXT"}

</button>

</div>

{loading && <div className="loading-spinner"></div>}

{error && <p className="error">{error}</p>}

{feedback && (

<div className="feedback-card">

<h2>FEEDBACK</h2>

<p>{feedback}</p>

<button

onClick={fetchSolution}

className="neon-button neon-button-purple"

>

VIEW SOLUTION

</button>

</div>

)}

{solution && (

<div className="solution-card">

<h2>OPTIMAL SOLUTION</h2>

<pre>{solution}</pre>

</div>

)}

</>

) : (

<div className="game-over">

{solvedCount >= 10 ? (

<>

<h2 className="game-over-title neon-text-green">MISSION COMPLETE!</h2>

<div className="final-stats">

<div className="stat-item">

<span className="stat-label">FINAL SCORE:</span>

<span className="stat-value neon-cyan">{totalScore}</span>

</div>

<div className="stat-item">

<span className="stat-label">CHALLENGES SOLVED:</span>

<span className="stat-value neon-purple">10/10</span>

</div>

</div>

</>

) : (

<>

<h2 className="game-over-title neon-text-red">TIME'S UP!</h2>

<div className="final-stats">

<div className="stat-item">

<span className="stat-label">FINAL SCORE:</span>

<span className="stat-value neon-cyan">{totalScore}</span>

</div>

<div className="stat-item">

<span className="stat-label">CHALLENGES SOLVED:</span>

<span className="stat-value neon-purple">{solvedCount}/10</span>

</div>

</div>

</>

)}

<div className="button-group">

<button

onClick={() => window.location.reload()}

className="neon-button neon-button-blue"

>

NEW GAME

</button>

<button

onClick={() => {

fetchQuestionScores();

setShowScores(!showScores);

}}

className="neon-button neon-button-magenta"

>

{showScores ? "HIDE SCORES" : "VIEW SCORES"}

</button>

</div>

{showScores && (

<div className="scores-container glassmorphic">

<h3>YOUR SCORES</h3>

<div className="scores-list">

{questionScores.length > 0 ? (

<table className="scores-table">

<thead>

<tr>

<th>Question</th>

<th>Score</th>

</tr>

</thead>

<tbody>

{questionScores.map((q) => (

<tr key={q.originalId}>

<td>#{q.displayNumber}</td>

<td className={

q.score >= 8 ? 'high-score' :

q.score >= 5 ? 'medium-score' : 'low-score'

}>

{q.score}/10

</td>

</tr>

))}

</tbody>

</table>

) : (

<p>No scores available</p>

)}

</div>

<div className="total-score-display">

TOTAL SCORE: <span className="neon-cyan">{totalScore}</span>

</div>

</div>

)}

</div>

)}

</div>

</div>

)}

</div>

);

}

export default App;

**Backend : Server.py**

from flask import Flask, request, jsonify

from flask\_cors import CORS

import logging

import ast

import requests

import re

from collections import defaultdict

import random

import uuid

import time

import json

from functools import lru\_cache

from tenacity import retry, stop\_after\_attempt, wait\_fixed

import subprocess

app = Flask(\_\_name\_\_)

CORS(app)

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

VALID\_COMPLEXITIES = {"O(1)", "O(log n)", "O(n)", "O(n log n)", "O(n^2)", "O(2^n)", "O(n!)"}

AI\_SERVICE\_URL = "http://127.0.0.1:11434/api/generate"

AI\_MODEL = "mistral"

AI\_TIMEOUT = 120

# Auto-start Ollama if not running

try:

    requests.get("http://127.0.0.1:11434", timeout=2)

    logger.info("Ollama is already running")

except:

    logger.info("Starting Ollama in background...")

    subprocess.Popen(["ollama", "serve"])

    time.sleep(3)  # Wait for initialization

# Combined single list of 30 questions

questions = [

    {"id": 1, "code": "def is\_positive(n):\n    if n > 0:\n        return True\n    return False"},

    {"id": 2, "code": "def max\_of\_two(a, b):\n    if a > b:\n        return a\n    else:\n        return b"},

    {"id": 3, "code": "def count\_vowels(s):\n    count = 0\n    for char in s:\n        if char in 'aeiouAEIOU':\n            count += 1\n    return count"},

    {"id": 4, "code": "def reverse\_words(s):\n    words = s.split()\n    words = words[::-1]\n    return ' '.join(words)"},

    {"id": 5, "code": "def factorial(n):\n    if n == 0:\n        return 1\n    result = 1\n    for i in range(1, n + 1):\n        result \*= i\n    return result"},

    {"id": 6, "code": "def fibonacci(n):\n    if n <= 1:\n        return n\n    a, b = 0, 1\n    for \_ in range(2, n + 1):\n        a, b = b, a + b\n    return b"},

    {"id": 7, "code": "def is\_palindrome(s):\n    s = s.lower()\n    for i in range(len(s) // 2):\n        if s[i] != s[len(s) - 1 - i]:\n            return False\n    return True"},

    {"id": 8, "code": "def binary\_search(arr, target):\n    left, right = 0, len(arr) - 1\n    while left <= right:\n        mid = (left + right) // 2\n        if arr[mid] == target:\n            return mid\n        elif arr[mid] < target:\n            left = mid + 1\n        else:\n            right = mid - 1\n    return -1"},

    {"id": 9, "code": "def sum\_of\_squares(n):\n    total = 0\n    for i in range(1, n + 1):\n        total += i \* i\n    return total"},

    {"id": 10, "code": "def shortest\_path(graph, start, end):\n    visited = set()\n    queue = [(start, [start])]\n    while queue:\n        node, path = queue.pop(0)\n        if node == end:\n            return path\n        if node not in visited:\n            visited.add(node)\n            for neighbor in graph[node]:\n                queue.append((neighbor, path + [neighbor]))\n    return None"},

    {"id": 11, "code": "def is\_even(n):\n    if n % 2 == 0:\n        return True\n    return False"},

    {"id": 12, "code": "def min\_of\_two(a, b):\n    if a < b:\n        return a\n    else:\n        return b"},

    {"id": 13, "code": "def count\_consonants(s):\n    count = 0\n    for char in s:\n        if char.isalpha() and char not in 'aeiouAEIOU':\n            count += 1\n    return count"},

    {"id": 14, "code": "def capitalize\_words(s):\n    words = s.split()\n    for i in range(len(words)):\n        words[i] = words[i].capitalize()\n    return ' '.join(words)"},

    {"id": 15, "code": "def power(base, exp):\n    result = 1\n    for \_ in range(exp):\n        result \*= base\n    return result"},

    {"id": 16, "code": "def tribonacci(n):\n    if n <= 1:\n        return 0\n    if n == 2:\n        return 1\n    a, b, c = 0, 0, 1\n    for \_ in range(3, n + 1):\n        a, b, c = b, c, a + b + c\n    return c"},

    {"id": 17, "code": "def is\_anagram(s1, s2):\n    s1 = s1.lower().replace(' ', '')\n    s2 = s2.lower().replace(' ', '')\n    if len(s1) != len(s2):\n        return False\n    for char in s1:\n        if char not in s2:\n            return False\n    return True"},

    {"id": 18, "code": "def linear\_search(arr, target):\n    for i in range(len(arr)):\n        if arr[i] == target:\n            return i\n    return -1"},

    {"id": 19, "code": "def product\_of\_numbers(n):\n    result = 1\n    for i in range(1, n + 1):\n        result \*= i\n    return result"},

    {"id": 20, "code": "def longest\_path(graph, start, end):\n    visited = set()\n    queue = [(start, [start])]\n    longest = None\n    while queue:\n        node, path = queue.pop(0)\n        if node == end and (longest is None or len(path) > len(longest)):\n            longest = path\n        if node not in visited:\n            visited.add(node)\n            for neighbor in graph[node]:\n                queue.append((neighbor, path + [neighbor]))\n    return longest"},

    {"id": 21, "code": "def is\_negative(n):\n    if n < 0:\n        return True\n    return False"},

    {"id": 22, "code": "def avg\_of\_two(a, b):\n    total = a + b\n    return total / 2"},

    {"id": 23, "code": "def count\_chars(s):\n    count = 0\n    for \_ in s:\n        count += 1\n    return count"},

    {"id": 24, "code": "def swap\_case(s):\n    result = ''\n    for char in s:\n        if char.isupper():\n            result += char.lower()\n        else:\n            result += char.upper()\n    return result"},

    {"id": 25, "code": "def sum\_up\_to(n):\n    total = 0\n    for i in range(1, n + 1):\n        total += i\n    return total"},

    {"id": 26, "code": "def lucas(n):\n    if n == 0:\n        return 2\n    if n == 1:\n        return 1\n    a, b = 2, 1\n    for \_ in range(2, n + 1):\n        a, b = b, a + b\n    return b"},

    {"id": 27, "code": "def is\_sorted(arr):\n    for i in range(len(arr) - 1):\n        if arr[i] > arr[i + 1]:\n            return False\n    return True"},

    {"id": 28, "code": "def find\_max(arr):\n    max\_val = arr[0]\n    for val in arr:\n        if val > max\_val:\n            max\_val = val\n    return max\_val"},

    {"id": 29, "code": "def sum\_of\_evens(n):\n    total = 0\n    for i in range(2, n + 1, 2):\n        total += i\n    return total"},

    {"id": 30, "code": "def depth\_first\_search(graph, start, end):\n    visited = set()\n    stack = [(start, [start])]\n    while stack:\n        node, path = stack.pop()\n        if node == end:\n            return path\n        if node not in visited:\n            visited.add(node)\n            for neighbor in graph[node]:\n                stack.append((neighbor, path + [neighbor]))\n    return None"}

]

test\_cases = {

    1: [((5,), True), ((-3,), False), ((0,), False), ((10\*\*6,), True), ((-10\*\*6,), False)],

    2: [((3, 7), 7), ((10, 2), 10), ((-1, -5), -1)],

    3: [(("hello",), 2), (("aeiou",), 5), (("xyz",), 0)],

    4: [(("hello world",), "world hello"), (("a b c",), "c b a"), (("single",), "single")],

    5: [((0,), 1), ((3,), 6), ((5,), 120)],

    6: [((1,), 1), ((3,), 2), ((5,), 5)],

    7: [(("radar",), True), (("hello",), False), (("Aba",), True)],

    8: [(([1, 2, 3, 4], 3), 2), (([1, 5, 9], 6), -1), (([], 1), -1)],

    9: [((3,), 14), ((4,), 30), ((1,), 1)],

    10: [(({"A": ["B"], "B": ["C"], "C": []}, "A", "C"), ["A", "B", "C"]), (({"A": ["B"], "B": []}, "A", "C"), None)],

    11: [((4,), True), ((3,), False), ((0,), True)],

    12: [((3, 7), 3), ((10, 2), 2), ((-1, -5), -5)],

    13: [(("hello",), 3), (("aeiou",), 0), (("xyz",), 3)],

    14: [(("hello world",), "Hello World"), (("a b",), "A B"), (("test",), "Test")],

    15: [((2, 3), 8), ((3, 2), 9), ((5, 0), 1)],

    16: [((1,), 0), ((3,), 1), ((5,), 4)],

    17: [(("listen", "silent"), True), (("hello", "world"), False), (("abc", "cba"), True)],

    18: [(([1, 2, 3], 2), 1), (([4, 5, 6], 7), -1), (([], 1), -1)],

    19: [((3,), 6), ((4,), 24), ((1,), 1)],

    20: [(({"A": ["B"], "B": ["C"], "C": []}, "A", "C"), ["A", "B", "C"]), (({"A": ["B", "C"], "B": ["C"], "C": []}, "A", "C"), ["A", "B", "C"])],

    21: [((-5,), True), ((3,), False), ((0,), False)],

    22: [((2, 4), 3.0), ((1, 5), 3.0), ((-2, -4), -3.0)],

    23: [(("hello",), 5), (("abc",), 3), (("",), 0)],

    24: [(("Hello",), "hELLO"), (("AbC",), "aBc"), (("x",), "X")],

    25: [((3,), 6), ((5,), 15), ((1,), 1)],

    26: [((0,), 2), ((1,), 1), ((4,), 4)],

    27: [(([1, 2, 3],), True), (([3, 1, 2],), False), (([],), True)],

    28: [(([1, 5, 3],), 5), (([-1, -2, -3],), -1), (([0],), 0)],

    29: [((4,), 6), ((6,), 12), ((2,), 2)],

    30: [(({"A": ["B"], "B": ["C"], "C": []}, "A", "C"), ["A", "B", "C"]), (({"A": ["B"], "B": []}, "A", "C"), None)]

}

players = {}

current\_round = str(uuid.uuid4())

@retry(stop=stop\_after\_attempt(3), wait=wait\_fixed(2))

def call\_ai\_service(prompt):

    """Call AI service with robust error handling and logging"""

    try:

        logger.debug(f"Sending to AI: {prompt[:100]}...")  # Log truncated prompt

        response = requests.post(

            AI\_SERVICE\_URL,

            json={

                "model": AI\_MODEL,

                "prompt": prompt,

                "stream": False,

                "options": {"temperature": 0.7}  # Added for better responses

            },

            timeout=AI\_TIMEOUT

        )

        # Debug raw response

        logger.debug(f"Raw AI response: {response.status\_code} {response.text[:200]}...")

        response.raise\_for\_status()

        data = response.json()

        # Validate response structure

        if not isinstance(data, dict) or "response" not in data:

            raise ValueError("Invalid AI response format")

        ai\_response = data["response"].strip()

        logger.debug(f"Received valid AI response: {ai\_response[:100]}...")

        return ai\_response

    except requests.exceptions.ConnectionError:

        logger.warning("Ollama connection failed - attempting to start...")

        subprocess.Popen(["ollama", "serve"])

        time.sleep(5)

        raise  # Will retry due to @retry

    except json.JSONDecodeError as e:

        logger.error(f"AI response JSON decode failed: {str(e)}")

        raise

    except requests.exceptions.RequestException as e:

        logger.error(f"AI connection failed: {str(e)}")

        raise

    except Exception as e:

        logger.error(f"Unexpected AI error: {str(e)}")

        raise

@lru\_cache(maxsize=1000)

def get\_complexity(code):

    try:

        prompt = f"""

        Analyze the time complexity of the following Python code and return ONLY the Big-O notation (e.g., O(1), O(n), O(n^2), etc.).

        Do not include any explanations or additional text.

        Code:

        ```python

        {code}

        ```

        Return format:

        Complexity: <O-notation>

        """

        ai\_response = call\_ai\_service(prompt)

        match = re.search(r"Complexity:\s\*(O\([^)]+\))", ai\_response)

        if match and match.group(1) in VALID\_COMPLEXITIES:

            return match.group(1)

        else:

            return "O(n)"

    except Exception as e:

        logger.error(f"AI complexity analysis failed: {str(e)}")

        return "O(n)"

@app.route("/register", methods=["POST"])

def register():

    data = request.get\_json()

    username = data.get("username", "").strip()

    if not username:

        return jsonify({"error": "Username required"}), 400

    if username in players:

        return jsonify({"error": "Username taken"}), 400

    # Create a copy of all questions and shuffle them

    all\_questions = questions.copy()

    random.shuffle(all\_questions)

    # Select first 10 questions for this player

    player\_questions = all\_questions[:10]

    players[username] = {

        "session\_id": current\_round,

        "score": 0,

        "questions\_answered": 0,

        "answered\_ids": set(),

        "attempted\_scores": {},

        "current\_set": player\_questions,

        "start\_time": time.time()

    }

    return jsonify({

        "session\_id": current\_round,

        "time\_limit": 900,

        "total\_questions": 10,

        "status": "registered"

    })

@app.route("/get\_challenge", methods=["POST"])

def get\_challenge():

    try:

        data = request.get\_json()

        username = data.get("username", "").strip()

        if not username or username not in players:

            return jsonify({"error": "Invalid session"}), 401

        player = players[username]

        # Only complete after 10 questions

        if len(player["answered\_ids"]) >= 10:

            return jsonify({

                "completed": True,

                "final\_score": player["score"],

                "progress": {

                    "answered": 10,

                    "total": 10

                }

            })

        unanswered = [q for q in player['current\_set']

                     if q["id"] not in player["answered\_ids"]]

        if not unanswered:

            return jsonify({"error": "No challenges available"}), 404

        next\_challenge = unanswered[0]

        return jsonify({

            "challenge": {

                "id": next\_challenge["id"],

                "code": next\_challenge["code"]

            },

            "progress": {

                "answered": len(player["answered\_ids"]),

                "total": 10

            },

            "time\_remaining": max(0, 900 - (time.time() - player["start\_time"]))

        })

    except Exception as e:

        logger.error(f"Challenge error: {str(e)}")

        return jsonify({"error": "Internal server error"}), 500

@app.route("/get\_all\_questions", methods=["POST"])

def get\_all\_questions():

    data = request.get\_json()

    username = data.get("username", "").strip()

    if not username or username not in players:

        return jsonify({"error": "Invalid session"}), 401

    player = players[username]

    # Create a list of all questions in order (1-30) with scores

    all\_questions = []

    for q in sorted(questions, key=lambda x: x["id"]):

        attempted = q["id"] in player["answered\_ids"]

        score = player["attempted\_scores"].get(q["id"], 0) if attempted else 0

        all\_questions.append({

            "id": q["id"],

            "code": q["code"],

            "score": score,

            "attempted": attempted

        })

    return jsonify({

        "questions": all\_questions,

        "status": "success"

    })

@app.route("/evaluate", methods=["POST"])

def evaluate():

    try:

        data = request.get\_json()

        username = data.get("username", "").strip()

        code = data.get("code", "").strip()

        challenge\_code = data.get("challenge\_code", "").strip()

        challenge\_id = data.get("challenge\_id")

        if username not in players:

            return jsonify({"error": "Invalid session"}), 400

        player = players[username]

        response\_data = {

            "feedback": "",

            "score": 0,

            "total\_score": player["score"],

            "is\_completed": len(player["answered\_ids"]) >= 10

        }

        if code.strip() == challenge\_code.strip():

            if challenge\_id not in player["answered\_ids"]:

                player["score"] += 5

                player["questions\_answered"] += 1

                player["answered\_ids"].add(challenge\_id)

                player["attempted\_scores"][challenge\_id] = 5

                response\_data["total\_score"] = player["score"]

            response\_data.update({

                "feedback": "score: 5\n⚠️ No changes made - You submitted the original challenge code verbatim (5/10 for base correctness)",

                "score": 5

            })

            return jsonify(response\_data)

        try:

            ast.parse(code)

        except SyntaxError as e:

            response\_data.update({

                "feedback": "score: 0\nSyntax Error: " + str(e),

                "score": 0

            })

            return jsonify(response\_data)

        challenge\_func = \_extract\_function\_name(challenge\_code)

        user\_func = \_extract\_function\_name(code)

        if not user\_func or user\_func != challenge\_func:

            response\_data.update({

                "feedback": f"score: 0\nFunction name must be '{challenge\_func}'",

                "score": 0

            })

            return jsonify(response\_data)

        test\_case\_key = challenge\_id

        if test\_case\_key not in test\_cases:

            return jsonify({"error": "Invalid challenge"}), 400

        try:

            namespace = {}

            exec(compile(code, "<string>", "exec"), namespace)

            user\_func = namespace[challenge\_func]

            for args, expected in test\_cases[test\_case\_key]:

                result = user\_func(\*args)

                if result != expected:

                    response\_data.update({

                        "feedback": f"score: 0\nTest failed for input {args}",

                        "score": 0

                    })

                    return jsonify(response\_data)

            ai\_opt\_code = \_get\_ai\_optimized\_code(challenge\_code)

            user\_complexity = get\_complexity(code)

            challenge\_complexity = get\_complexity(challenge\_code)

            score, feedback = \_calculate\_score(

                code, challenge\_code, ai\_opt\_code,

                user\_complexity, challenge\_complexity

            )

            if challenge\_id not in player["answered\_ids"]:

                player["score"] += score

                player["questions\_answered"] += 1

                player["answered\_ids"].add(challenge\_id)

                player["attempted\_scores"][challenge\_id] = score

            response\_data.update({

                "feedback": feedback,

                "score": score,

                "total\_score": player["score"],

                "is\_completed": len(player["answered\_ids"]) >= 10

            })

            return jsonify(response\_data)

        except Exception as e:

            response\_data.update({

                "feedback": f"score: 0\nRuntime error: {str(e)}",

                "score": 0

            })

            return jsonify(response\_data)

    except Exception as e:

        logger.error(f"Evaluation endpoint error: {str(e)}")

        return jsonify({"error": "Internal server error"}), 500

@app.route("/get\_solution", methods=["POST"])

def get\_solution():

    try:

        data = request.get\_json()

        code = data.get("code", "").strip()

        username = data.get("username", "").strip()

        if not code:

            return jsonify({"error": "No code provided"}), 400

        if username not in players:

            return jsonify({"error": "Invalid session"}), 400

        prompt = f"""Return ONLY the raw optimized Python code with:

        - No explanations

        - No introductory text

        - No markdown formatting

        Original code:

        ```python

        {code}

        ```"""

        response = requests.post(

            AI\_SERVICE\_URL,

            json={

                "model": AI\_MODEL,

                "prompt": prompt,

                "stream": False,

                "options": {"temperature": 0.3}

            },

            timeout=30

        )

        response.raise\_for\_status()

        ai\_response = response.json().get("response", "").strip()

        solution = re.sub(r'```python|```', '', ai\_response).strip()

        return jsonify({

            "solution": solution if solution else code,

            "status": "success"

        })

    except requests.exceptions.RequestException as e:

        logger.error(f"AI service error: {str(e)}")

        return jsonify({

            "solution": "# Error generating solution\n" + code,

            "status": "error"

        }), 500

    except Exception as e:

        logger.error(f"Unexpected error: {str(e)}")

        return jsonify({

            "solution": "# Error generating solution\n" + code,

            "status": "error"

        }), 500

def \_extract\_function\_name(code):

    try:

        return next(

            (node.name for node in ast.walk(ast.parse(code))

             if isinstance(node, ast.FunctionDef)), None

        )

    except Exception:

        return None

def \_get\_ai\_optimized\_code(code):

    prompt = """Optimize this Python code with these STRICT REQUIREMENTS:

1. You MUST return a different implementation than the original

2. The optimized version MUST have better time or space complexity

3. If the original is already optimal (like O(1)), improve readability or conciseness

4. Return ONLY the raw optimized code with NO explanations

5. Never return identical code to the original

Original code:

```python

{code}

```"""

    max\_attempts = 3

    attempt = 0

    while attempt < max\_attempts:

        try:

            ai\_response = call\_ai\_service(prompt)

            optimized\_code = re.sub(r'```python|```', '', ai\_response).strip()

            # Verify the optimized code is different from original

            if optimized\_code and optimized\_code != code.strip():

                return optimized\_code

            attempt += 1

            logger.warning(f"AI returned same code as original (attempt {attempt})")

            time.sleep(1)  # Brief delay before retry

        except Exception as e:

            logger.error(f"Optimization error on attempt {attempt}: {str(e)}")

            attempt += 1

    # Fallback: Return a slightly modified version if AI fails

    logger.warning("Using fallback optimization after max attempts")

    return \_fallback\_optimize(code)

def \_fallback\_optimize(code):

    """Fallback optimization when AI fails to provide a proper optimized version"""

    try:

        # Try to make minimal safe changes

        tree = ast.parse(code)

        # Example: Change variable names or simple refactors

        for node in ast.walk(tree):

            if isinstance(node, ast.Name) and not isinstance(node.ctx, ast.Load):

                node.id = f"opt\_{node.id}"

        return ast.unparse(tree)

    except:

        # If all else fails, return original but with a warning

        logger.error("Fallback optimization failed - returning original")

        return code

def \_calculate\_score(user\_code, challenge\_code, ai\_code, user\_complexity, challenge\_complexity):

    # First check if user submitted the original challenge code

    if user\_code.strip() == challenge\_code.strip():

        return 5, "score: 5\n🔍 Submitted original code (5/10 for correctness - try optimizing for higher scores!)"

    complexity\_order = {"O(1)": 1, "O(log n)": 2, "O(n)": 3,

                       "O(n log n)": 4, "O(n^2)": 5, "O(2^n)": 6, "O(n!)": 7}

    user\_rank = complexity\_order.get(user\_complexity, 4)

    challenge\_rank = complexity\_order.get(challenge\_complexity, 4)

    # Check if AI code is actually different and better

    ai\_is\_optimized = (ai\_code.strip() != challenge\_code.strip())

    ai\_is\_better = False

    if ai\_is\_optimized:

        ai\_complexity = get\_complexity(ai\_code)

        ai\_rank = complexity\_order.get(ai\_complexity, 4)

        ai\_is\_better = ai\_rank < challenge\_rank

    if ai\_is\_optimized and ai\_is\_better and user\_code.strip() == ai\_code.strip():

        return 10, "SCORE: 10\n✅ Perfect! Matches AI-optimized solution!"

    elif user\_rank < challenge\_rank:

        return 9, f"SCORE: 9\n🌟 Excellent! Improved complexity ({user\_complexity} vs original {challenge\_complexity})"

    elif user\_rank == challenge\_rank:

        if len(user\_code.splitlines()) < len(challenge\_code.splitlines()):

            return 8, f"SCORE: 8\n💡 Great! Same complexity but more concise ({user\_complexity})"

        return 7, f"SCORE: 7\n👍 Good! Correct with same complexity ({user\_complexity})"

    else:

        return 6, f"SCORE: 6\n⚠️ Works but less efficient ({user\_complexity} vs original {challenge\_complexity})"

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(host="0.0.0.0", port=5001, debug=True)

**Github Link:** [**https://github.com/codeczars/GenAI.git**](https://github.com/codeczars/GenAI.git)

# Steps for Code Execution

## Setup Environment

**Prerequisites**

Ensure you have the following installed:

1. **Ollama** - Download and install from [Ollama's official site](https://ollama.ai/).
2. **Mistral Model** - Install it using the command:
   * ollama pull mistral
3. **Python** (latest version recommended)
4. **Node.js & npm** (latest version recommended)

**Backend Setup**

1. Open a command prompt (CMD) and run the following command to start Ollama:
   * ollama serve
2. Navigate to the Backend folder:
   * cd backend
3. Open server.py in a code editor.
4. Run the backend server using:
   * python server.py

**Frontend Setup**

1. Navigate to the Frontend folder:
   * cd frontend
2. Open the src directory and locate App.js.
3. Install dependencies by running:
   * npm install
4. Start the frontend application using:
   * npm start

**Notes**

* Make sure **Ollama is running** before starting the backend.
* If you face any issues, check the logs in server.py and fix any dependency errors.
* You can modify frontend styles and logic inside the src folder.

## Running AI Model (Ollama)

* + Ensure Ollama is running:
  + ollama serve

## Playing the Game

* + Open the web interface
  + Log in and select a challenge
  + Optimize the provided code and submit
  + Receive AI feedback and score

# Input & Output

## Input:

## 

## 

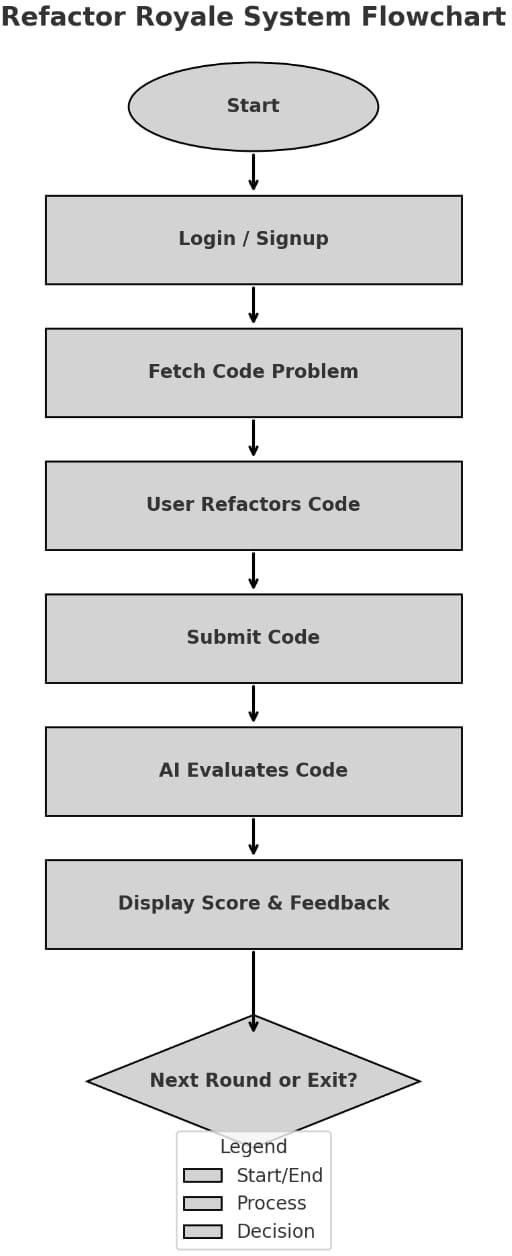
* + Pre-written inefficient code snippets (e.g., sorting algorithms, loops, recursive functions)

## Output:

## 

* + Optimized code with AI-generated feedback on execution time, readability, and maintainability.
  + Score based on code improvement and best practices.

**Flowchart**

****

**Video Demo**

A demo video showcasing the gameplay, AI evaluations, and scoring system is available at:

**Video Link 1** : <https://drive.google.com/file/d/1s3zrgcRIV03sTJb_CwdtLQ25UlSDp1tL/view?usp=drivesdk>

**Video Link 2: it shows time’s up:**

<https://drive.google.com/file/d/1sSyWlNjwUBzWCHu7GMiz1lFsi88sRuS8/view?usp=drivesdk>

# GitHub Repository

All source code, documentation, and additional details can be accessed at:

<https://github.com/codeczars/GenAI.git>

# Additional Technical Details

## Frontend Development

* + React: UI components using hooks (useState, useEffect)
  + Vite: Fast build tool with hot module replacement
  + CodeMirror: Python syntax highlighting for live editing
  + React Confetti: Celebration effects for achievements
  + CSS3: Neon glow effects, grid/flexbox layouts

## Backend Development

* + Flask: REST API endpoints
  + Flask-CORS: Secure frontend-backend communication
  + Python AST: Code structure validation
  + Subprocess: Running AI service (Ollama)

## AI & Optimization Engine

* + Ollama: Local LLM runner with Mistral 7B
  + Tenacity: Retry mechanism for AI calls
  + LRU Cache: Performance optimization

## Game Logic & Evaluation

* + Challenge Pool: 30 Python problems with varying complexities
  + Scoring System: Dynamic scoring (5-10 points) based on optimization
  + Time Tracking: 15-minute challenge limit
  + Progress System: Tracks solved challenges with persistent scores

## Hardware Requirements

* + **Minimum:** CPU with AVX support (for Ollama)

### Recommended:

* + - NVIDIA GPU (for faster AI inference)
    - 16GB RAM (for smooth Ollama operation).

# Conclusion

Refactor Royale successfully gamifies code optimization, making it an engaging way for developers to improve their skills. By integrating AI for evaluation and feedback, the game provides an interactive learning experience, enhancing understanding of coding best practices. The project showcases the potential of AI-driven education tools, making code optimization accessible and enjoyable.