PhishGuard Applied Research

Authors: Roko Mladinić, Daniella Namuli, Simeon Markov, Zed Minabowan & Kristiano Mizher

Table of contents

Introduction 2
Role 1: cybersecurity specialist A (The attacker) 2
Research Task 1: Phishing Taxonomy 3
Research Task 2: Social Engineering Psychology 3
Research Task 3: Evasion Techniques 4
Role 2: cybersecurity specialist B (The Defender) 6
Research Task 1: Forensic Indicators 6
Research Task 2: Educational Best Practices 6
Role 3: Game Developer 8
Research Task 1: Gamification Loops 8
Research Task 2: Progression Systems 10
Research Task 3: Feedback Timing 13
Role 4: Software Engineer 16
Research Task 1: Frontend Framework Comparison: 16
Research Task 2: Backend & Database Options 18
Research Task 3: Deployment & Collaboration 19

Introduction

In this document, we state the research we did before beginning our group project. We made five roles and divided them among each other. We then did research on the specific roles we received and used this research to formulate our game.

Role 1: Cybersecurity Specialist A (The attacker)

I did research using the library / deskresearch according to the CMD research methods. This is where I use existing resources to conduct my research. I had three research tasks and used several websites to answer these research tasks. All websites used are linked in the references.

Research Task 1: Phishing Taxonomy

Phishing attacks come in different forms, each designed to trick users in specific ways. Understanding these types is important to create realistic and varied game levels.

- **Spear Phishing** targets a specific person using personal or contextual information.
- **Whaling (CEO Fraud)** targets employees by impersonating high-level executives.
- **Clone Phishing** copies legitimate emails and replaces links or attachments with malicious ones.
- **Link Manipulation** disguises malicious links to appear safe or trustworthy.

Deliverable: A list of 10 distinct attack scenarios to be used as game levels.

1. **Fake IT Support Email (Spear Phishing)**
   An email sent to an employee claiming their password is expiring and asking them to log in immediately.
2. **CEO Urgent Payment Request (Whaling)**
   An email pretending to be from the CEO requesting an urgent bank transfer.
3. **HR Document Update (Clone Phishing)**
   A copied HR email where the original attachment is replaced with a malicious file.
4. **Package Delivery Notification (Link Manipulation)**
   A fake delivery email containing a shortened link to “track” a package.
5. **Microsoft Account Alert (Spear Phishing)**
   A personalized email warning about suspicious login activity.
6. **Invoice Reminder from Supplier (Clone Phishing)**
   A fake invoice sent using a real supplier’s name and email style.
7. **Prize or Gift Card Win (Greed-based Phishing)**
   An email claiming the user has won a reward and must click to claim it.
8. **Password Reset Confirmation (Link Manipulation)**
   A login button that redirects to a fake website with a lookalike domain.
9. **Team Meeting Change (Spear Phishing)**
   An email pretending to be from a colleague with a malicious calendar link.
10. **Security Policy Violation Warning (Whaling)**
    An email threatening disciplinary action unless the user responds quickly.

Research Task 2: Social Engineering Psychology

Social engineering attacks rely on manipulating human emotions rather than exploiting technical weaknesses. Hackers use psychological triggers to influence users into acting without thinking critically.

The most common triggers are:

- Urgency: Creating pressure to act quickly.

- Fear: Threatening negative consequences.
- Greed: Offering rewards or benefits.
- Curiosity: Encouraging clicks through mystery or interest.

Deliverable: Short Guide "How to Write a Convincing Fake Email"

To create a convincing phishing email, attackers usually follow these steps:

1. **Impersonate a trusted source**
   Use the name, logo, or writing style of a well-known company or authority figure.
2. **Create urgency or fear**
   Include phrases like "Immediate action required" or "Your account will be suspended."
3. **Keep the message short and direct**
   Phishing emails are often brief to avoid detailed scrutiny.
4. **Include a clear call to action**
   Examples include "Click here," "Verify now," or "Download the document."
5. **Avoid obvious mistakes in advanced attacks**
   More sophisticated phishing emails have corrected grammar and realistic formatting.

This guide ensures the game's phishing emails are realistic and reflect real-world attacker behaviour.

Research Task 3: Evasion Techniques

Attackers use evasion techniques to hide malicious links and bypass both user suspicion and basic security checks. These techniques make phishing emails harder to detect, especially for inexperienced users.

Deliverable: Deceptive Techniques for "Hard Mode" Levels

1. **URL Shorteners**
   Using shortened links to hide the real destination website.
2. **Typosquatting**
   Creating fake domains that closely resemble real ones (e.g, g00gle.com).
3. **Subdomain Tricks**
   Using misleading subdomains such as paypal.login-secure.com.
4. **Image-Based Links**
   Embedding malicious links inside images instead of text.
5. **Button-Based Redirection**
   Buttons labelled as "Secure Login" that lead to fake websites.
6. **HTTPS Abuse**
   Using HTTPS to appear legitimate, even though the site is malicious.

7. **Email Address Spoofing**

   Displaying a trusted sender name while using a different email address.

These techniques will be implemented in advanced game levels to increase difficulty and improve user awareness.

By studying these techniques, I gained insight into how attackers design phishing emails that appear legitimate. This knowledge will be directly used to create realistic and challenging game scenarios.

Role 2: cybersecurity specialist B (The Defender)

Research Task 1: Forensic Indicators

Checklist of Red Flags

Based on the literature study, the following indicators are commonly used to identify phishing emails:

- **Suspicious sender address**
  The sender's domain does not match the claimed organization or contains small spelling changes.
- **Generic greeting**
  Messages that start with phrases like "Dear user" instead of using a real name.
- **Urgent or threatening language**
  Emails that pressure the user with deadlines, warnings about account suspension, or fear-based messages.
- **Unexpected attachments or links**
  Files or links that the user was not expecting to receive.
- **Mismatched URLs**
  The visible link's text does not match the actual destination of the link.
- **Requests for sensitive information**
  Emails that ask for passwords, verification codes, or personal information.

This checklist forms the core decision logic of the game. Each email scenario in PhishGuard is built around one or more of these indicators.

Research Task 2: Educational Best Practices

Feedback Pop-up Text for Non-Technical Users

The literature study shows that non-technical users learn best when feedback is:

- Immediate
- Short
- Focused on one mistake at a time
- Written in simple and clear language

Examples of feedback text used in PhishGuard include:

- **Incorrect action example**
  “This email was dangerous. The sender address looks official, but the domain name is slightly misspelled. Attackers often use this trick to look trustworthy.”
- **Correct action example**
  “Good job. You noticed the urgent language and the suspicious link. Real companies rarely ask for sensitive information by email.”

These explanations are meant to teach users what to look for next time, instead of only telling them that they made a mistake.

Research Task 3: Data Points for Analysis

Dashboard Metrics

Based on cybersecurity training literature, the following metrics are the most useful for analyzing user awareness:

1. **Correct identification rate**
   The percentage of emails that were correctly identified as safe or malicious.
2. **Time to decision**
   How long a user takes to decide whether to flag or accept an email.
3. **False positive rate**
   The number of safe emails that were incorrectly flagged as malicious.
4. **Missed threat rate**
   The number of malicious emails that were incorrectly accepted.
5. **Threat category weaknesses**
   User performance per phishing type, such as fake invoices or credential theft.

These metrics help organizations identify weak points in user behavior and the project’s needs.

Role 3: Game Developer

🎮 Role 4: Game Developer Focus: Game Mechanics, Logic & Engagement. Now that the Media Designer handles the looks, this role focuses on the rules and the fun.

After searching for methods of research on cmdmethods.nl, i decided to start with “Benchmark Creation”. It mentions that it is necessary to scout for products that are related to my design goal. I googled some IT education websites, such as Udemy and Brilliant.

Udemy offers cybersecurity awareness training for employees, and it costs 15€, while our solution will be free. On the other hand, Brilliant is a “freemium” website, offering some training (not related to cybersecurity) and then blocking you behind a paywall. This is not a good option for people that want to learn something and have fun at the same time.

After not getting much information from that CMD method, I switched to “**Best, good & bad practices”,** which suggests comparing our product idea to an already existing product. To research my first task, I decided to look up some famous games that have core loops and take notes of how they balance risk and reward.

Research Task 1: Gamification Loops

Research "Core Loops" in gaming. How do you balance risk vs. reward? (e.g., "Flagging correctly gives +10 points, but flagging a safe email loses -20 points").

Deliverable: A rule-set document defining how the scoring system works.

1. Papers, Please (The "Investigator" Loop)

This is the most relevant game for your project. The player acts as a border control officer, checking documents against a rulebook.

- **The Core Loop:** Receive Documents → Inspect for Discrepancies (Dates, Names, Stamps) → Make Binary Decision (Approve/Deny) → Receive Consequence (Salary or Penalty).
- **Why it works:** It turns "spot the difference" into a tense job simulation. Every move that you make can change the outcome of the story.
- **What to adapt for PhishGuard:**
    - **The "Rulebook":** In *Papers, Please*, you have a book you must constantly check. In *PhishGuard*, you could have a "Security Handbook" button the user can check if they are unsure.
    - **The Penalties:** It doesn't just say "Wrong." It gives you a "Citation" slip. You could replicate this by printing a "Security Violation Ticket" on the screen when the user fails.

2. Reigns (The "Rapid Fire" Loop)

This game uses a "Tinder-style" mechanic where you rule a kingdom by swiping left or right on cards.

- **The Core Loop:** Read Scenario Card → Swipe Left or Right (Binary Choice) → Immediate Stat Change (Health/Money goes up or down) → Next Card.
- **Why it works:** It is incredibly fast. There is no complex menu; just yes or no.
- **What to adapt for PhishGuard:**
    - **The Input:** Instead of clicking small buttons, allow the user to swipe the email left (Delete) or right (Reply). This makes the physical action of "cleaning the inbox" satisfying.
    - **The Stats:** *Reigns* has 4 distinct health bars (Church, People, Army, Money). You could track 4 distinct metrics: *System Health, Trust, Network Speed, and Budget.*

https://thomasj49.sg-host.com/reigns-review-474750/#:~:text=FAQ,alliances%2C or even abrupt endings.

3. Fruit Ninja (The "Reflex" Loop)

A classic arcade game where you slice fruit but avoid bombs.

- **The Core Loop:** Object Spawns → Identify (Fruit vs. Bomb) → Action (Slice or Wait) → Score.
- **Why it works:** It relies on **Pattern Recognition**. You see a "Bomb" and your brain instantly says "Don't touch."
- **What to adapt for PhishGuard:**
    - **The "Bomb":** Treat "Ransomware" emails like Bombs in Fruit Ninja. If the user touches them, it's instant Game Over (or massive damage).
    - **The Pacing:** Start slow (one email every 10 seconds) and speed up until emails are flying in every 2 seconds. This creates the "Panic" that leads to real-world security mistakes.

https://elena-berman.medium.com/mechanics-dynamics-and-aesthetics-of-fruit-ninja-23aaa72fd6c7

While “Papers, Please” resembles “PhishGuard” the most, it is still necessary to view and compare other games with similar core loops. All of these games have something distinctive that can be mentioned:

Game	The Mechanic	The Feeling	What PhishGuard takes from it
Papers, Please	Cross-referencing data	"I need to be accurate."	The logic of checking the "Sender" against the "Domain."
Reigns	Swiping Left/Right	"I need to be decisive."	The interface (Accept vs. Reject).
Fruit Ninja	Identifying targets	"I need to be fast."	The visual distinction between "Safe" (Fruit) and "Virus" (Bomb).

My solution to the "PhishGuard" Core Loop

Based on these three games, this is my proposal for “PhishGuard”:

1. **Spawn:** Email appears with a countdown timer.
2. **Observation:** Player scans for *Key Indicators* (bad logo, typo).
3. **Action:** Player clicks a possible discrepancy to check, or presses reply to mark it safe.
4. **Feedback:**
    a. *If Correct:* Green flash, +100 Points, satisfying "Ding" sound.
    b. *If Wrong:* Screen "glitches" (simulating a virus), -20 Health, and a popup explains the mistake.
5. **Progression:** Speed increases by 5% for the next email.

This video showcases several "sorting" style games, which is helpful to see how other designers make the act of "organizing" (or in our case, filtering emails) feel satisfying rather than like a chore: 10 Satisfying Organizing & Sorting Games You Need to Try!

Research Task 2: Progression Systems

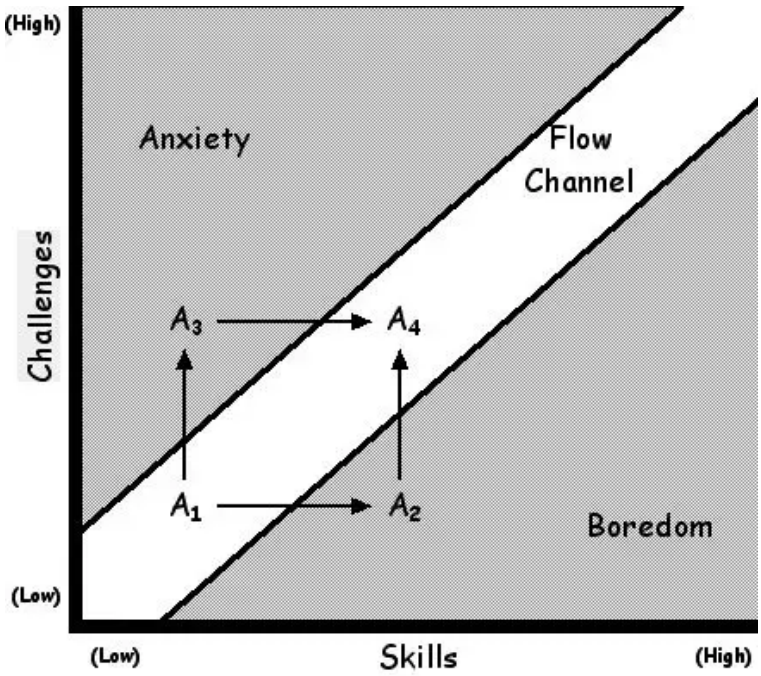
Research how to scale difficulty. How do we move the user from "Beginner" to "Expert"? (e.g., decreasing time limits, removing visual hints).

Deliverable: A "Level Design" plan (Level 1 vs. Level 5 parameters).

For the second task, I decided to go with “Literature study” as my CMD Method. Since I am looking for the rules and theories of how to balance a game and the psychological principles that successful game designers use, this is the most logical option.

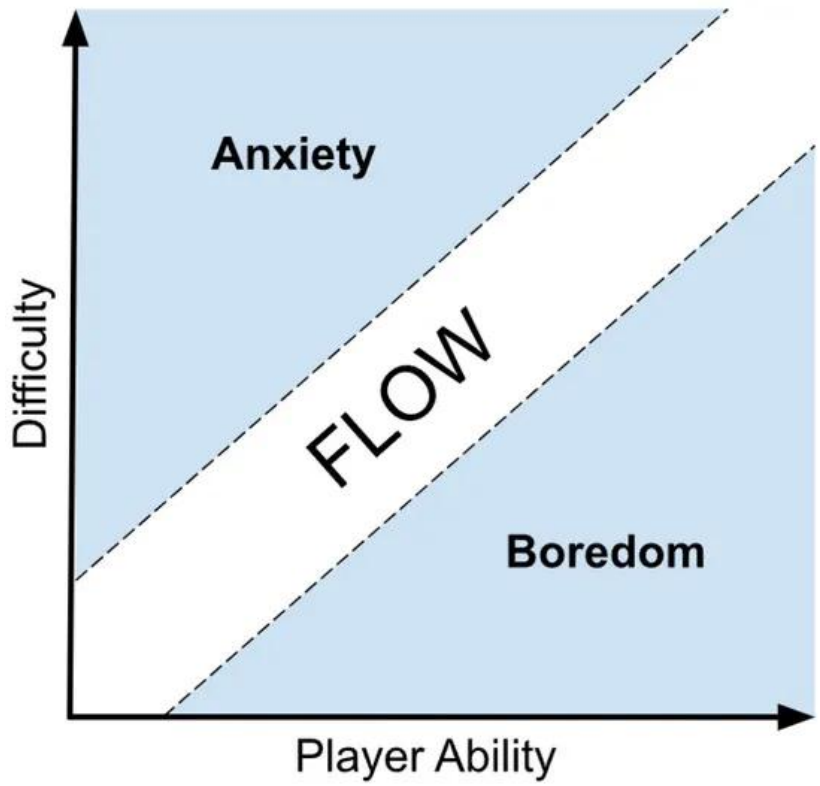
The first piece of information I found was about “The Flow Channel”.

The Concept: If a game is too hard, the player feels **Anxiety**. If it is too easy, they feel **Boredom**. You want them in the middle channel: **Flow**.



From Flow: The Psychology of Optimal Experience
by Mihaly Csikszentmihalyi (page 74)

The Psychology of Optimal Experience, by Mihály Csikszentmihály, https://ricardo-valerio.medium.com/make-it-difficult-not-punishing-7198334573b8



The Flow Channel diagram, https://www.mdpi.com/2813-2084/3/2/12

An interesting paragraph on the Game Developer website resonated with me, and I'll likely implement this approach after finishing my research:

‼

One way to do this, write down a list of challenges that you want the player to overcome in the game. Then, write down the skills that will be needed to complete these challenges. Take the naïve design approach, where you program the game to where the player has a challenge, and the necessary skill to complete that challenge. Pay no mind to whether the game is too easy or too hard. Once you have all the challenges and skills in the game the way you like, then go back and tweak the areas using the oscillation and reward method.

https://www.gamedeveloper.com/design/understanding-the-flow-channel-in-game-design

Another important aspect will be the difficulty of the game. It is important to manage multiple variables in order to keep the game interesting, which is why skill and difficulty progression come hand in hand. A good example of this method is previously mentioned Papers, Please and Overcooked.

Overcooked uses different variables as "levers" that developers adjust. As the game progresses, orders become more complex, more cooking stations are required, and food preparation steps increase.

My plan for the variables

These are the variables I plan on using for this game:

1. ⏳ Time Variables (The Pressure) These are the easiest to code and the most effective way to create stress (and therefore errors).

• **Spawn Rate:** How many seconds wait between emails? ◦ Easy: 10 seconds (User waits for mail). ◦ Hard: 0.5 seconds (Spam flood).

• **Decision Timer:** How long does the user have to act before the email "auto-accepts" or they get a penalty? ◦ Easy: Unlimited time. ◦ Hard: 5 seconds.

• **Day Length:** How long does a "Level" last? ◦ Variable: 60 seconds vs. 180 seconds (Endurance test).

2. 📨 Content Variables (The Deception) These variables control the "quality" of the fake emails. You can assign a "Complexity Score" to each email in your database.

• **Typos & Grammar:** ◦ Easy: "Dear Cuztomer" (Obvious). ◦ Hard: Perfect English, professional tone.

• **Sender Address Quality:** ◦ Easy: support@gmaill.com (Public domain). ◦ Medium: admin@paypa1.com (Typosquatting). ◦ Hard: security@paypal-support.com (Look-alike domain).

• **Emotional Trigger:** ◦ Variable: None vs. High Fear ("Your account is banned!") vs. High Greed ("You won $500!"). High emotion usually leads to faster, worse decisions.

3. 👁️ Visual Variables (The Distraction) These variables mess with the player's ability to see the clues clearly.

• **UI "Glitch" Intensity:** ◦ Variable: Screen shake or static noise effect when the system health gets low.

• **Helpful Highlights:** ◦ Easy: Suspicious words are highlighted in yellow (Training wheels). ◦ Hard: No highlights at all.

• **Desktop Clutter:** ◦ Hard: Pop-up windows (fake ads or "System Update" reminders) cover the email, forcing the player to close them to read the text.

4. ⚖️ Game Mechanic Variables (The Consequences) These control the rules of winning and losing.

• **Health (Trust) Pool:** ◦ Easy: 5 Strikes allowed. ◦ Hard: 1 Strike = Game Over (Sudden Death).

• **Streak Multiplier:** ◦ Variable: 1x, 2x points. (Does the game reward getting 10 right in a row? This encourages the "Flow" state).

• **Feedback Timing:** ◦ Easy: Immediate pop-up ("Wrong!"). ◦ Hard: Delayed feedback (You only find out you failed at the end of the day).

Research Task 3: Feedback Timing

Research the difference between "Immediate Feedback" (interrupting the game) and "Delayed Feedback" (end-of-level summary). Which is better for learning?

Deliverable: A decision on the game flow (e.g., "Does the game stop when you fail, or continue until the end?").

Given the project's five-week timeline, surveys and game testing (including A/B testing) aren't feasible. I'll use the "Literature study" research method instead.

The first option for giving feedback back to the player is **Immediate Feedback (Interruption).**

How it works: User clicks "Accept" on a virus, then the Game instantly pauses and it displays the text: "WRONG! That was a virus."

The Science: This is best for **Novices** and **Correction**. It prevents the user from forming bad habits. If they don't know why they failed, they can't learn.

Game Examples: *Duolingo* (buzzes immediately if you miss a word), *Super Mario* (you die the second you touch a goomba, spike, flame…).

Pros: High learning retention for beginners.

Cons: Breaks the "Flow." Can be annoying if it happens too often.

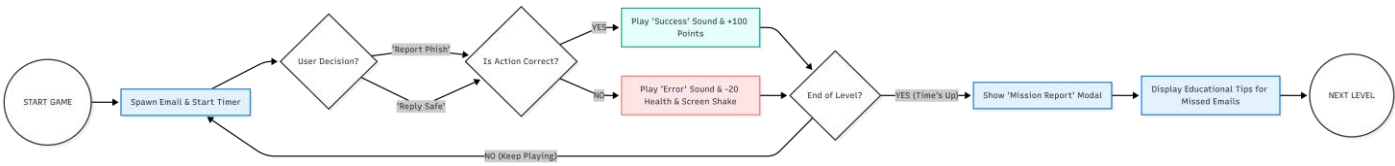
The second option is by having **Delayed Feedback / Summary:**

- **How it works:** User processes 10 emails. They made 3 mistakes, but the game kept going. At the end, a screen says: "You failed 3 times. Here is what you missed."
- **The Science:** This is best for **Experts** and **Fluency**. It trains the user to trust their gut under pressure.
- **Game Examples:** *Guitar Hero* (shows your % accuracy at the end of the song), *Civilization* (you see the result of your strategy turns later).
- **Pros:** Keeps the game exciting and fast. Simulates real life (you don't know you've been hacked until later).
- **Cons:** Users might forget *which* email they messed up by the time they see the result.

My solution to the feedback issue

I've decided that PhishGuard should use a hybrid model. The tutorial level will provide immediate feedback, then switch to delayed feedback once the player understands the game. This approach prevents breaking the "Flow Channel" too often, which can frustrate players and make them give up.

The game will also include subtle immediate feedback—such as animations or text pop-ups—that doesn't stop gameplay but provides helpful clues. At the end of each round, a daily report will show all the mistakes and how to fix them.



To visualize the game mechanics, I created a flowchart using Mermaid.live.

This flowchart illustrates the **Core Gameplay Loop**, demonstrating how the user's interaction with email triggers immediate system feedback (points/health) before cycling back to the next email or proceeding to the educational summary at the end of the level.

Role 4: Software Engineer

Focus: Technology Stack Selection & Architecture This role is the technical architect. Instead of just "using Python," they must find the best tool for the job.

Research Task 1: Frontend Framework Comparison

Compare lightweight web technologies suitable for a 5-week project. Should you use Vanilla JS, Vue.js, or React? What are the learning curves for each?

Deliverable: A recommendation on the frontend technology with a "Why I chose this" justification.

Research Task 2: Backend & Database Options

Research different backends (Node.js, Python, Go) and databases (SQL vs. NoSQL/JSON). Which one is easiest to set up for a simple scoreboard?

Deliverable: An architecture diagram showing how the frontend, backend, and database will connect.

Research Task 3: Deployment & Collaboration

Research how the team will code together. Will you use GitHub? How will you host the game online for the final presentation (e.g., Vercel, Netlify, Heroku, Render)?

Deliverable: A "DevOps" plan—a setup guide for the team to start coding.

Research Task 1: Frontend Framework Comparison:

1. FRONTEND FRAMEWORK: Vue.js 3 + Nuxt 3

Why This Choice? [1] [7]

Performance Advantages:

- 19% faster startup time vs React (critical for mobile users)
- 36% faster DOM manipulation (interactive quizzes respond instantly)
- Smaller initial JavaScript bundle (reduced bandwidth for global audiences)
- Better performance on lower-end devices (reaches underserved markets)

Developer Experience:

- HTML-based templates (easier to learn than JSX)
- Pinia state management (significantly simpler than Redux/Context API)

- Auto-imports (less boilerplate code)
- Composition API with <script setup> syntax (modern, intuitive)
- TypeScript support (first-class, not bolted-on)
- File-based routing (folder structure = application structure)

Team Productivity:

- Smaller learning curve = faster hiring
- Built-in development server with hot module reloading
- Excellent developer tools (Vue DevTools with timeline debugging)

Educational Clarity:

- Vue mirrors HTML structure (closer to traditional web development)
- Easier to teach programming concepts without abstraction layers
- Clear separation: templates, scripts, styles

Styling: Tailwind CSS + shadcn/ui

Why Tailwind CSS:

- Industry standard in 2025 (definitively replaced Bootstrap)
- Utility-first approach enables rapid, consistent design
- Zero CSS naming conflicts (eliminates "what should I call this class?" decisions)
- Design tokens built-in (color palette, spacing, sizing)
- Tree-shakeable (unused styles don't ship to production)
- Dark mode support built-in (modern user expectation)

Why shadcn/ui (React) or Headless UI (Vue):

- **Copy-paste components** - Not locked into library updates.
- **Accessibility guaranteed** - WCAG 2.1 compliance by default
- **Full customization** - modifying components without dealing with CSS overrides
- **Small bundle** - only components that are needed
- **Type-safe** - TypeScript-first component design
- **Educational value** - beginners see production-ready component patterns

Research Task 2: Backend & Database Options

2. BACKEND FRAMEWORK: Nitro (Nuxt Server) + Node.js

Why Unified Nuxt Full-Stack? [6]

Single Application Deployment:

- Frontend (Vue components) + Backend (API routes) = one Git repo
- One deployment command instead of two
- Environment variables managed once
- Single monitoring/logging system

Simpler Development:

- One Node.js version to manage
- One package manager (pnpm)
- One testing setup
- One CI/CD pipeline

3. DATABASE: PostgreSQL [3]

ACID Transactions:

- User progress update cannot partially fail (all-or-nothing guarantee)
- Quiz submission with score recording must be atomic
- Payment processing requires guarantees

Complex Queries:

- Aggregate user progress: "Show me lesson completion %, average quiz score, learning streak"
- Requires multi-table joins: Users → Lessons → Quiz_Responses → Scores

Analytics:

- Window functions for ranking (leaderboards)
- Aggregation functions for statistics
- Time-series analysis (learning over weeks/months)

Research Task 3: Deployment & Collaboration

4. DEPLOYMENT: Railway [2]

- npm run deploy or GitHub integration
- Automatic SSL certificates (no manual setup)
- PostgreSQL included (add via UI, provisioned instantly)
- Environment variables stored securely

CI/CD Pipeline: GitHub Actions [4] [5]

Why GitHub Actions?

- Native GitHub integration (no external tools)
- Free for public repos, generous free tier for private
- Marketplace with pre-built actions

Testing:

- **Vitest** - Fast unit testing (10x faster than Jest)
- **Playwright** - End-to-end testing (test real user flows)
- **Percy** - Visual regression testing (catch UI changes)

Layer	Technology	Why
Frontend Framework	Vue.js 3 + Nuxt 3	Faster, simpler, better DX
Styling	Tailwind CSS + shadcn/ui	Industry standard, zero conflicts
Type Safety	TypeScript	Shared frontend-backend types
State Management	Pinia	Simpler than Redux, excellent DevTools

Backend	Nitro (Nuxt server)	Unified fullstack, single deployment
Database	PostgreSQL	ACID guarantees, complex queries, analytics
ORM	Prisma	Type-safe, auto-generated types
Deployment	Railway	Fast, cheap, PostgreSQL included
Repository	Monorepo + Turborepo	Shared types, single source of truth
Package Manager	pnpm	Faster, efficient, monorepo support
Testing	Vitest + Playwright	Fast unit tests, real user flow E2E
CI/CD	GitHub Actions	Native GitHub, no external tools
Code Quality	ESLint + Prettier + TypeScript	Automated standards enforcement

References:

[01] Top 7 Frontend Frameworks for Web Development in 2025. https://webpinn.com/best-7-frontend-frameworks-that-accelerate-pace-of-web-development/

[02] Trang. chủ (2025). Railway Review 2025 – A Modern Deployment Platform for Developers, Startups, SMEs, and Students https://ikigaiteck.com/pages/railway-review-2025-modern-app-deployment-platform

[03] Hamza, Khan (2025). PostgreSQL vs. MongoDB in 2025: Which Database Should Power Your Next Project? https://dev.to/hamzakhan/postgresql-vs-mongodb-in-2025-which-database-should-power-your-next-project-2h97

[04] Learn continuous integration with GitHub Actions. https://learn.microsoft.com/en-us/training/modules/learn-continuous-integration-github-actions/

[05] Omar, Elhawary (2023). Building a full-stack TypeScript application with Turborepo. https://blog.logrocket.com/build-full-stack-typescript-application-turborepo/

[06] Arunangshu, Das (2025). Choosing the Right Node.js Framework: Options and Comparisons. https://arunangshudas.com/blog/choosing-node-js-framework-options-and-comparisons/

[07] Top Frontend Technologies, Tools, and Frameworks to Use in 2025. https://www.wedowebapps.com/frontend-technologies-tools-frameworks/

Role 1 References:

19 Types of Phishing Attacks with Examples | Fortinet

9 types of phishing attacks and how to identify them | CSO Online

What Are the Different Types of Phishing? | Trend Micro (US)

19 Most Common Types of Phishing Attacks in 2026 | UpGuard