**Truly**

**Screening Test/Interview Questions for Web Developer (Remote) Position**

**Instructions: Please proceed all your answers with the question you are answering.**

Use acronyms only after you've explained them.

Use correct spelling and grammar.

Be sure to write your name.

Candidate’s Name: **Andrew Reddin**

Interviewer’s Name: **Irene Johnson**

**Can you walk us through a recent web project you've worked on? What was your role, and what technologies did you use**?  
At Lexamica, I led the front-end implementation of an attorney referral platform, translating Figma designs into a production React/Next.js application. My role involved:

* Working closely with designers to implement pixel-perfect UI components
* Building reusable component libraries that matched design system specifications
* Creating responsive layouts and animations that enhanced user experience
* Establishing component documentation and style guidelines
* Implementing real-time data visualization dashboards from design mockups

Core Technologies:

* React 18 with Next.js 13 for server-side rendering
* TypeScript for type safety
* Zustand for state management
* TailwindCSS for styling
* React Query for data fetching
* Jest and React Testing Library for testing

**What’s your process for debugging and troubleshooting code? Can you give an example of a complex issue you've solved?**

My debugging process follows a systematic approach:

* I start with browser dev tools - React DevTools for component inspection and Chrome DevTools for network requests and performance
* I use TypeScript's type checking to catch issues early
* For complex state issues, Zustand DevTools helps trace state changes
* I implement logging at critical points using tools like LogRocket to actually view the process used by the user and trace the bug.

**How do you approach optimizing website performance? Can you share specific tools or techniques you use?**

For performance optimization, I focus on:

Key Metrics:

* Core Web Vitals, including: • Largest Contentful Paint (LCP): time to render largest content element • First Input Delay (FID): time before page responds to interaction • Cumulative Layout Shift (CLS): measures visual stability
* Time to First Byte (TTFB): time until first byte of page content received
* First Contentful Paint (FCP): time until first content appears

Tools & Techniques:

* Next.js Image optimization for faster loading
* Dynamic imports for code splitting
* React Suspense for smooth loading states
* Zustand for efficient state management
* Network optimization using: • Static Site Generation (SSG): pre-renders pages at build time Example: Legal documentation, firm profiles, FAQs that rarely change • Incremental Static Regeneration (ISR): regenerates static pages periodically Example: Referral listings, fee calculators updated hourly
* Chrome DevTools Performance tab
* Lighthouse audits
* Bundle analysis with @next/bundle-analyzer
* LogRocket for real user monitoring

**What is your experience with responsive web design? Can you explain how you ensure websites work across different devices and screen sizes?**

To ensure cross-device compatibility, I:

1. Design with mobile-first principles

* Start with mobile layouts
* Progressively enhance for larger screens
* Test breakpoints at common device sizes

1. Implement responsive components

* Flexible grid systems
* Dynamic font scaling
* Conditional rendering based on screen size
* Touch-friendly interaction areas

1. Rigorous testing

* Device lab testing on physical devices
* Browser dev tools for device simulation
* User session recordings via LogRocket
* Automated viewport testing

**Can you explain the difference between == and === in JavaScript? Why is it important?**

The key difference between == and === in JavaScript is type coercion. The loose equality operator (==) performs type coercion before comparison, meaning '5' == 5 returns true because JavaScript converts the string '5' to a number before comparing. In contrast, the strict equality operator (===) compares both value and type without conversion, so '5' === 5 returns false because a string is not strictly equal to a number. At Lexamica, we enforced strict equality (===) in our TypeScript codebase to prevent subtle bugs, especially when comparing API responses with expected values where type consistency was crucial for data validation.

**What’s your experience with version control, specifically Git? How do you handle branching and merging in a team environment?**

At Lexamica, we implemented a structured Git workflow where feature branches were created from our staging branch for individual development. Each developer worked locally, submitted pull requests with detailed descriptions, and required peer review before merging. Our staging environment served as an integration point where automated tests and QA were performed before production deployment. We protected main and staging branches, requiring approvals and passing CI/CD checks through GitHub Actions. To maintain a clean commit history, we used squash merging and comprehensive PR templates. We maintained regular deployment schedules, with staging deployments happening daily and production deployments weekly after thorough testing. This process helped us maintain code quality while enabling continuous delivery of features.

**How do you stay up-to-date with web development trends, frameworks, and best practices? Can you name a recent framework or technology you've learned or experimented with?**

I stay current with web development through daily coding practice, tech newsletters like JavaScript Weekly and the Next.js blog, following GitHub trending repositories, and contributing to open source projects.

Recently, I've deepened my expertise with React Query, implementing it at Lexamica to improve our data fetching patterns. The switch to React Query significantly reduced our API calls through automatic caching, simplified state management, and improved user experience with built-in loading and error states. This implementation resulted in smoother data updates and better performance across our attorney referral platform.

**Can you explain the concept of the DOM (Document Object Model) and how it interacts with JavaScript?**

The Document Object Model (DOM) is a programming interface that represents HTML documents as a tree structure of objects. Each HTML element becomes a node in this tree, which JavaScript can manipulate. When JavaScript interacts with the DOM, it can dynamically modify content, structure, and styling. For example, at Lexamica, we implemented a real-time filtering system for attorney listings - when users select specialization filters, JavaScript updates specific DOM nodes to show or hide relevant attorneys without a page refresh. The DOM also provides events (like clicks and form submissions) that JavaScript listens to, enabling interactive features like our dynamic sorting and filtering system.

**What is your experience with CSS preprocessors like Sass or LESS? How do they improve your workflow?**

While I'm proficient in CSS preprocessors like Sass and LESS, at Lexamica we primarily used TailwindCSS. Sass enhances CSS workflow by allowing nested styles, similar to HTML structure, making code more organized and maintainable. Instead of writing multiple selectors, you can nest them:

| .header {  background: blue;  .nav {  padding: 10px;  &:hover { background: lightblue; }  } } |
| --- |

At Lexamica, we chose TailwindCSS for its utility-first approach and built-in PostCSS processing, which provided:

* Component-based styling
* Custom variables for theming
* Responsive utilities
* @apply directives
* Built-in vendor prefixing

This improved development speed while maintaining consistent styling.

This stack improved development speed and maintained consistent styling across our application. When we needed custom styles, we used PostCSS plugins to extend Tailwind's functionality.

**How do you manage state in a front-end JavaScript framework, such as React, Vue, or Angular? Can you give an example?**

At Lexamica, I used Zustand for state management because it offers a streamlined approach to handling application state in React. It uses a simple hook pattern that allows components to subscribe to specific state slices, automatically optimizing performance by only updating components affected by state changes. Zustand's architecture provides centralized state stores with straightforward set/get operations, built-in TypeScript support, and middleware for state persistence. Its lightweight nature significantly reduced boilerplate code compared to alternatives like Redux, while maintaining powerful features like async actions support and DevTools integration. This made our state management more maintainable and performant across the entire application.

| // Store definition const useReferralStore = create((set) => ({  referrals: [],  loading: false,  setReferrals: (data) => set({ referrals: data }),  fetchReferrals: async () => {  set({ loading: true });  const data = await api.getReferrals();  set({ referrals: data, loading: false });  } }));  // Component usage function ReferralList() {  const { referrals, loading, fetchReferrals } = useReferralStore();    useEffect(() => {  fetchReferrals();  }, []);   return loading ? <Spinner /> : <List referrals={referrals} />; } |
| --- |

**What back-end technologies and frameworks are you most familiar with (e.g., Node.js, Express, Django, Ruby on Rails)? How do you integrate them with front-end code?**

In my role at Lexamica, I worked extensively with Node.js and Express, integrating them with our React frontend through RESTful APIs. We structured our Express server with middleware layers for authentication, request validation, and error handling. For database operations, we used MongoDB with Mongoose for schema validation and TypeScript for type safety. The frontend utilized React Query for efficient data fetching, caching, and state management, significantly reducing API calls. Our API responses were role-based - administrators received full data access while referral managers got filtered data based on their permissions. A key challenge was optimizing database queries when referral managers requested filtered data, which initially caused performance issues. We solved this by implementing database indexing and moving the filtering logic to the database layer rather than filtering in memory.

**What are some best practices for securing a web application, especially in terms of handling user data and authentication?**

For web application security at Lexamica, we implemented several key practices:

Authentication:

* JSON Web Token (JWT) tokens with secure expiration
* Password hashing with bcrypt
* One-time codes for secure login
* Session management and secure cookies

Data Protection:

* Hypertext Transfer Protocol Secure (HTTPS)/Transport Layer Security (TLS) encryption
* Field-level encryption for sensitive data
* Input validation and sanitization
* Protection against Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) attacks
* Environment variables for sensitive data
* Rate limiting on Application Programming Interface (API) endpoints

Database Security:

* Prepared statements to prevent Structured Query Language (SQL) injection
* Encrypted sensitive fields
* Regular security audits
* Limited database access permissions

**Describe a time when you had to collaborate with non-technical stakeholders (e.g., designers, project managers). How did you ensure effective communication and meet their needs?**

At Lexamica, I collaborated closely with product managers and legal experts to implement a matching algorithm for referrals. I translated complex business rules about case types and jurisdictions into technical requirements. During stakeholder meetings, I used flowcharts to visualize the matching logic, making it easier for non-technical team members to understand and refine the algorithm. This collaborative approach helped us create an efficient system that correctly matched referrals to law firms based on their expertise and geographical coverage, improving the referral success rate by 40%.

**Will you be able to devote 20-35 hours of your time per week to Truly Beauty?**Yes, I am able to commit 20-35 hours per week, though I would prefer and am available for a full-time position if that opportunity exists. I'm enthusiastic about bringing my full expertise and dedication to Truly Beauty.

**What do you understand by the word “Privacy and code of conduct?**

In software development, privacy and code of conduct are crucial ethical principles. Privacy means protecting user data through practices like field-level encryption, secure data handling, and strict access controls. At Lexamica, we handled sensitive legal data, so privacy was paramount in our development process.

Code of conduct refers to professional behavior standards, including:

* Respectful communication with team members
* Ethical handling of sensitive information
* Following security protocols
* Maintaining confidentiality
* Adhering to company policies

For example, when working with attorney referral data, we ensured all team members understood and followed strict data protection protocols.

**How much will you request per hour if you are hired?**

Based on my experience as a Lead Front-End Engineer with extensive React, Next.js, and TypeScript expertise, my hourly rate range is $65-80. This reflects my 5+ years of experience delivering enterprise-scale applications and leading development teams.