

Frontend Engineer – Live Coding Assessment

Overview

This assessment evaluates your ability to build a functional multi-feature application that demonstrates:

- Clean code architecture and organisation
- API integration and data management
- User interface design and user experience
- Problem-solving and decision-making skills

Duration: 3-4 hours

Focus: Quality over quantity, build a working application with 2-3 well-implemented features rather than trying to complete everything

Technology Stack


Choose the platform you're most comfortable with:

Web:

- React, Vue, Angular, Next.js, or vanilla JavaScript
- Any CSS framework (Tailwind, Material-UI, Bootstrap, etc.)

Mobile:

- React Native or Flutter

 **Tip:** We recommend React or Next.js for web developers, Flutter for mobile developers.

Project: E-Commerce Application

You will build a feature-rich application that combines e-commerce functionality. The application should demonstrate your ability to integrate multiple APIs, design intuitive interfaces, and implement complex user interactions.

API Resources

Base URL: <https://dummyjson.com>

- **Products:** [/products](#) - Browse products, search, filter
- **Carts:** [/carts](#) - Shopping cart data
- **Users:** [/users](#) - User profiles and information
- **Posts:** [/posts](#) - Social posts by users

 **Full Documentation:** [Docs - DummyJSON - Free Fake REST API for Placeholder JSON Data](#)

Required Features

Feature 1: Product Catalogue (REQUIRED)

Build a product browsing experience with the following:

Product List Page

- Display products in a grid or list layout
- Show: product image, title, price, rating, and brand
- Load at least 30 products (pagination or infinite scroll)
- Make products clickable to view details

Search and Filters

Implement at least **2 of the following**:

- **Search bar** – Filter by product name
- **Category filter** – Filter by product category
- **Price range** – Min/max price slider or input
- **Rating filter** – Minimum rating selector

💡 Filters should work together when multiple are active

Product Detail Page

When clicking a product, show:

- Large product image(s)
- Full description
- Price, stock status, and rating
- Customer reviews
- "Add to Cart" button

UI Requirements

- Loading spinner while data loads
 - "No results found" message when filters return no products
 - Error message if API fails
 - Smooth navigation between list and detail views
 - Responsive design (works on mobile and desktop)
-

Feature 2: Shopping Cart (REQUIRED)

Build a functional shopping cart system:

Cart Functionality

- Add products to the cart from the product detail page
- Update quantity (increase/decrease)
- Remove items from cart

- Calculate totals:
 - Subtotal (sum of items)
 - Tax (10% of subtotal)
 - Grand Total

Cart Page

Create a dedicated cart page showing:

- All cart items with images, names, and prices
- Quantity controls for each item
- Pricing breakdown (subtotal, tax, total)
- "Empty cart" state when no items
- Item count badge in navigation

Cart Persistence

- Keep cart data during the session
- **Bonus:** Save cart to localStorage so it persists after page refresh

User Experience

- Visual confirmation when adding to cart (e.g., notification, animation)
 - Confirm before removing items ("Are you sure?")
 - Disable actions on empty cart
-

Feature 3: User Profiles (CHOOSE ONE APPROACH)

Option A: User List + Profile Pages (Simpler)

- Display a list of users from the API
- Click a user to see their profile page with:
 - Basic info (name, email, phone, address)
 - Their posts (from Posts API)
 - Their cart history (from Carts API)

Option B: Mock Authentication (More Advanced)

- Create a simple login page
- Let users "sign in" by selecting a user from the API
- Show personalised profile with their data
- Display their specific cart and posts

💡 Choose the option that interests you more and fits your time budget

Technical Requirements

1. Navigation

Implement clear navigation between:

- Home/Products page
- Cart page
- User List or Profile page
- Product Detail page

Web: Use React Router, Vue Router, or similar

Mobile: Use Navigation Stack/Controller

2. State Management


Choose an appropriate solution for your stack:

Web:

- React Context API (simple projects)
- Zustand (lightweight, recommended for this size)
- Redux Toolkit (if you prefer Redux)

Mobile:

- React Native: Context API or Redux
- Flutter: Provider, Riverpod, or Bloc

 Briefly explain your choice in the README

3. Code Organisation

Structure your code logically:

```
None
src/
├─ components/      # Reusable UI components
├─ pages/           # Page-level components
├─ services/        # API calls and data fetching
├─ hooks/           # Custom React hooks (if applicable)
├─ utils/           # Helper functions
└─ styles/          # CSS/styling files
```

Best Practices:

- Keep components small and focused (single responsibility)
- Extract reusable logic into custom hooks or utilities
- Use consistent naming conventions
- Add comments for complex logic

4. API Integration

Create a clean API service layer:

```
JavaScript
// Example structure
services/
  api.js           // Base API configuration
  products.js      // Product-related API calls
  carts.js         // Cart-related API calls
  users.js         // User-related API calls
```

Handle API states properly:

- Loading states (show spinners)
- Success states (display data)
- Error states (show error messages)

5. Error Handling

Implement user-friendly error handling:

- Network errors: "Unable to load products. Please check your connection."
 - Empty results: "No products found matching your filters."
 - API errors: "Something went wrong. Please try again."
-

Bonus Features (Optional – Pick 1 if time permits)

Only attempt these if you've completed all required features with quality:

Bonus 1: Search with Debouncing

- Add 300-500ms delay to search API calls while the user types
- Show "Searching..." indicator
- Cancel the previous request when a new search starts

Bonus 2: Dark Mode Toggle

- Add a light/dark theme switcher
- Save preference to localStorage
- Style all components for both themes

Bonus 3: Enhanced UI Animations

- Smooth page transitions
- Add-to-cart animation
- Skeleton loaders instead of spinners

- Hover effects and micro-interactions

Bonus 4: Advanced Filters

- Multiple sort options (price: low-high, high-low, rating, name)
 - Show number of results
 - Quick filter chips/tags
 - "Clear all filters" button
-

Submission Requirements

What to Submit

1. Source Code

- Complete project with all files
- Clean, organised folder structure
- No `node_modules` or build files committed

2. README.md (Important!) Include:

- Brief project description (2-3 sentences)
- Technologies used and why
- Setup instructions (step-by-step)
- Features completed checklist
- Known issues or limitations
- Time spent (approximate)
- Assumptions made

3. Dependencies File

- `package.json` (Web)
- `pubspec.yaml` (Flutter)

4. Screenshots (Optional but recommended)

- 3-5 screenshots showing main features
- Or a short video demo (2-3 minutes)

How to Submit

1. Push code to the provided GitHub repository
2. Create a Pull Request and add reviewer: shiva.aryal@intuji.com
3. Email confirmation to the hiring team
4. Include the repository link in the email (optional)

README Template

None

E-Commerce Application

Brief description of your application.

Technologies Used

- [Framework/Library] - Why I chose it
- [State Management] - Why I chose it
- [Styling Solution] - Why I chose it

Features Completed

- [x] Product listing with search and filters
- [x] Product detail page
- [x] Shopping cart with add/remove/update
- [x] Cart persistence
- [] User profiles (not completed)

Setup Instructions

1. Clone the repository
2. Install dependencies: `npm install`
3. Start dev server: `npm run dev`
4. Open browser to `http://localhost:3000`

Time Spent

Approximately 3.5 hours

Known Issues

- Filter combinations could be optimized
- Cart doesn't sync across tabs

Assumptions Made

- No real authentication required
- Tax rate fixed at 10%
- Using mock data for cart operations

Time Management Guide

Total Time: 3-4 hours

Suggested Timeline

Time	Activity
0:00 - 0:30	Set up project, install dependencies, plan architecture
0:30 - 1:45	Feature 1: Product listing, search/filters, detail page
1:45 - 2:45	Feature 2: Shopping cart functionality and UI
2:45 - 3:30	Feature 3: User profiles (simplified version)
3:30 - 4:00	Polish UI, test features, write README, prepare submission

Time-Saving Tips

1. **Start with what you know best** - Build on your strengths first
 2. **Use a starter template** - Create React App, Vite, Next.js starter
 3. **Don't over-engineer** - Simple solutions are perfectly fine
 4. **Copy-paste styling** - Focus on functionality over perfect design
 5. **Test as you go** - Catch bugs early rather than at the end
 6. **Document while coding** - Update README as you complete features
-

Evaluation Criteria

Your submission will be evaluated on:

Criteria	Weight	What We Look For
Functionality	35%	Features work as expected, no critical bugs
Code Quality	30%	Clean, readable, well-organised code
UI/UX	20%	Intuitive interface, good user experience
Technical Decisions	15%	Appropriate tool choices, good architecture

What "Good" Looks Like

Functionality:

- All required features work correctly
- The app doesn't crash with normal usage
- API integration works reliably

Code Quality:

- Easy to read and understand
- Logical file organisation
- Reusable components
- Consistent formatting

UI/UX:

- Clean, professional appearance
- Responsive layout
- Clear feedback for user actions
- Intuitive navigation

Technical Decisions:

- Appropriate state management
 - Efficient API calls
 - Reasonable component structure
-

Common Pitfalls to Avoid

Don't:

- Try to implement every feature perfectly
- Spend 2 hours on styling
- Over-complicate state management
- Forgot to handle loading/error states
- Skip the README
- Commit node_modules or .env files

Do:

- Focus on core functionality first
 - Use simple, clean styling
 - Handle edge cases (empty states, errors)
 - Write clear, commented code
 - Test your app before submitting
 - Follow the submission checklist
-

Pre-Submission Checklist

Before you submit, verify:

- ☐ App runs without errors (`npm start` or equivalent)
 - ☐ All required features are working
 - ☐ README.md is complete with setup instructions
 - ☐ Code is properly formatted and organised
 - ☐ Navigation between pages works
 - ☐ Loading states show during API calls
 - ☐ Error messages appear when the API fails
 - ☐ No console errors in the browser
 - ☐ Git repository is clean (no build files)
 - ☐ Someone else could run your app following your README
-

API Quick Reference

Common Endpoints You'll Need

JavaScript

// Get all products

GET <https://dummyjson.com/products>

// Search products

GET <https://dummyjson.com/products/search?q=phone>

// Get single product

GET <https://dummyjson.com/products/1>

// Get product categories

GET <https://dummyjson.com/products/categories>

// Get products by category

GET <https://dummyjson.com/products/category/smartphones>

// Get all users

GET <https://dummyjson.com/users>

// Get single user

GET <https://dummyjson.com/users/1>

// Get user's posts

GET <https://dummyjson.com/posts/user/1>

// Get user's carts

GET <https://dummyjson.com/carts/user/1>

API Response Examples

Product:

JSON

```
{
  "id": 1,
  "title": "iPhone 13",
  "price": 999,
  "rating": 4.7,
  "category": "smartphones",
  "thumbnail": "https://...",
  "images": ["https://..."],
  "stock": 50
}
```

User:

JSON

```
{
  "id": 1,
  "firstName": "John",
  "lastName": "Doe",
  "email": "john@example.com",
  "phone": "+1234567890",
  "image": "https://..."
}
```

Support & Questions

During the Assessment

If you encounter:

- **API issues:** Document the problem and use mock data
- **Technical blockers:** Document assumptions made
- **Unclear requirements:** Make reasonable assumptions and note them in README

After Submission

Be prepared to:

- Walk through your code
 - Explain technical decisions
 - Discuss alternative approaches
 - Answer questions about implementation
-

Final Tips for Success

1. **Read everything first** - Understand all requirements before coding
 2. **Plan your time** - Don't spend 3 hours on one feature
 3. **Start simple** - Get basic version working, then enhance
 4. **Commit frequently** - Save your progress regularly
 5. **Test thoroughly** - Click through every feature before submitting
 6. **Communicate clearly** - Your README is your voice
-

Good Luck! 🚀

This is your opportunity to showcase your skills as a frontend engineer. We're looking for:

- **Problem-solving ability** - How you approach challenges
- **Code craftsmanship** - Quality over quantity
- **User empathy** - Building for real users
- **Communication** - Through code and documentation

Remember: **2 features done well > 3 features done poorly**

If you have questions during the assessment, contact your hiring coordinator.

Frequently Asked Questions

General Questions

Q: What if I can't complete all features in time?

A: That's completely fine and expected! Focus on doing 2 features really well rather than rushing through all 3. Quality beats quantity. Document what you completed and what you'd add with more time in your README. We'd rather see clean, working code for 2 features than buggy code for 3.

Q: How will my submission be evaluated?

A: We evaluate based on four main areas: Functionality (35%), Code Quality (30%), UI/UX (20%), and Technical Decisions (15%). We're looking for working features, clean code, good user experience, and smart technology choices.

Q: Can I submit even if I didn't finish everything?

A: Yes! Submit what you have. A well-executed partial solution is valuable. Just be clear in your README about what's complete and what's missing.

Technology & Tools

Q: Can I use UI component libraries?

A: Yes! Feel free to use:

- **Web:** Material-UI, Ant Design, Tailwind, Chakra UI, Bootstrap, shadcn/ui
- **Mobile:** Material Design, Cupertino, React Native Paper

However, you should still demonstrate some custom styling to show your design skills. Don't just use default components for everything.

Q: Can I use TypeScript?

A: Absolutely! TypeScript is encouraged and shows strong engineering practices. If you're comfortable with it, use it.

Q: Can I use a starter template or boilerplate?

A: Yes! You can use:

- Create React App, Vite, Next.js starter
- Flutter starter template
- Any framework CLI tool

Just ensure the application logic, component structure, and features are your own work.

Q: What about state management libraries?

A: Use whatever you're comfortable with:

- Simple apps: React Context API, useState
- Medium complexity: Zustand (recommended), Jotai
- Full featured: Redux Toolkit, MobX

Pick what fits the project size. Don't over-engineer with Redux if Context API works fine.

Code Quality

Q: Should I write tests?

A: Tests are **not required** for this assessment. Focus on functionality and code quality first. If you finish early and want to showcase testing skills, feel free to add a few key tests, but it's not expected.

Q: How much should I comment my code?

A: Comment on complex logic and non-obvious decisions. Your code should be self-documenting through clear naming. Add comments where "why" isn't obvious from reading the code.





Q: What coding style should I follow?

A: Use consistent formatting (Prettier is fine). Follow common conventions for your chosen framework. The main thing is consistency throughout your codebase.

Design & UI

Q: Should I focus more on functionality or design?

A: Aim for a balance – **70% functionality, 30% design.**

-  Clean, functional UI that works well
-  Proper spacing, readable text, logical layout
-  Don't spend 2 hours perfecting animations
-  Don't build a pixel-perfect design system

A working app with decent UI beats a beautiful mockup that doesn't function.

Q: Do I need to make it pixel-perfect?

A: No! We're not evaluating pixel perfection. Your app should look clean and professional, but don't stress over perfect alignment or exact spacing.

Q: Should I design for mobile responsiveness?

A: If building for the web, yes - your app should work reasonably well on desktop and mobile viewports. You don't need to support every screen size perfectly, just ensure it's usable on common sizes.

API & Data

Q: What if the DummyJSON API is down or slow?

A:

1. Implement proper error handling (you'll be evaluated on this)
2. Document the issue in your README
3. Optional: Create a mock data file as a fallback
4. Show how you'd handle API failures gracefully

Q: Can I cache API responses?

A: Yes! Caching is smart. You can:

- Cache in component state
- Use localStorage for persistence
- Implement a simple cache layer in your API service

Just keep it simple - don't build a complex caching system.

Q: The API doesn't support DELETE/UPDATE for carts. What should I do?

A: That's expected! Simulate these operations locally:

- Manage cart state in your frontend
- Treat it as if the API calls succeed
- Document this assumption in your README

Prohibited & Allowed

Q: Can I use AI coding assistants (GitHub Copilot, ChatGPT, Claude, etc.)?

A: NO – Strictly prohibited. This assessment evaluates YOUR skills and problem-solving ability. Using AI assistants will result in immediate disqualification. We need to see your authentic work to assess your fit for this role.

Q: Can I search Google/Stack Overflow for syntax help?

A: Yes! Looking up documentation, syntax, or how to use a specific API is completely normal and allowed. We're testing your ability to build an application, not memorise syntax. Examples of acceptable searches:

- "How to use React Router"
- "Array filter method JavaScript"
- "Tailwind CSS grid layout"

Q: Can I look at the framework/library documentation?

A: Yes, absolutely! Official documentation is always allowed:

- React docs, Vue docs, Flutter docs
- npm package documentation
- CSS framework documentation

Q: Can I reuse code from my previous projects?

A: Yes, if it's YOUR code. Don't copy others' projects, but reusing your own utility functions or component patterns is fine.

Submission

Q: What format should my README be in?

A: Markdown (.md) format. Use the template provided in this document as a starting point.

Q: Should I deploy the application?

A: Not required, but if you can quickly deploy to Vercel, Netlify, or similar (5–10 minutes), it's a nice bonus. Don't spend significant time on deployment.

Q: Can I submit a video walkthrough instead of written documentation?

A: Video is a nice addition, but doesn't replace the written README. Include both if you have time.

Q: What if I find a bug after submitting?

A: Mention it in your README under "Known Issues" if you notice before the review. If you discover it after submission, you can email a quick note to the interviewer, but don't resubmit unless explicitly asked.

Time Management

Q: What if I'm running out of time?

A:

1. Stop adding features – focus on making what you have work well
2. Test what you've built
3. Write your README (critical – don't skip!)
4. Submit what you have.

A well-documented partial submission is better than a rushed complete one.

Q: Can I take breaks during the assessment?

A: Yes! This isn't a timed exam. Take breaks as needed. The 3–4 hours is the total coding time.

Q: What if something comes up and I need more time?

A: Contact your hiring coordinator as soon as possible. They'll advise on next steps.

Technical Issues

Q: What if I encounter a bug I can't solve?

A:

1. Document it in your README under "Known Issues"
2. Explain what you tried to fix it
3. Move on to other features
4. We evaluate problem-solving, not perfection

Q: My development server won't start. What should I do?

A:

1. Check Node version (use LTS version, 20+ recommended)
2. Delete `node_modules` and reinstall: `rm -rf node_modules && npm install`
3. Clear cache: `npm cache clean --force`
4. Document the issue and your resolution steps

Q: What if I'm stuck on a particular problem?

A:

1. Spend 15-20 minutes trying to solve it
 2. If still stuck, simplify the approach or move to another feature
 3. Document what you tried in your README
 4. We value seeing your thought process
-

Still Have Questions?

If your question isn't answered here:

- **Before starting:** Contact your hiring coordinator
- **During assessment:** Make a reasonable assumption and document it in your README
- **After submission:** Bring it up in the follow-up discussion