# 🎨 Componentes e Padrões de Usabilidade para Página de Features - Detalhamento Técnico

Este documento fornece uma análise aprofundada de cada componente, incluindo implementações práticas, considerações de acessibilidade e exemplos interativos.

## 1. Hero Section com Seletor de Persona

### Implementação Detalhada

```
interface PersonaCardProps {
  avatar: string;
  title: string;
  description: string;
  onClick: () => void;
}

const PersonaCard: React.FC<PersonaCardProps> = ({
  avatar,
  title,
  description,
  onClick,
}) => (
  <div
    role="button"
    aria-label={`Selecionar persona ${title}`}
    tabIndex={0}
    className="persona-card"
    onClick={onClick}
    onKeyDown={(e) => e.key === "Enter" && onClick()}
  >
    <img
      src={avatar}
      alt={`Avatar ${title}`}
      aria-hidden="true"
      className="avatar"
    />
    <h3 className="persona-title">{title}</h3>
    <p className="persona-description">{description}</p>
  </div>
);

// Implementação do HeroSection
const HeroSection = () => {
  const [selectedPersona, setSelectedPersona] = useState("small");

  return (
```

```
    <section aria-labelledby="hero-heading" className="hero-section">
      <h1 id="hero-heading">Recursos para seu perfil de produtor</h1>
      <div className="persona-selector">
        {personas.map((persona) => (
          <PersonaCard
            key={persona.id}
            {...persona}
            onClick={() => setSelectedPersona(persona.id)}
          />
        ))}
      </div>
      <PersonaFeatures persona={selectedPersona} />
    </section>
  );
};
```

### Considerações de Acessibilidade

- Foco no primeiro elemento interativo
- Navegação por teclado (Tab/Shift+Tab)
- Leitores de tela anunciam mudanças de estado
- Contraste mínimo 4.5:1 para textos
- Zoom seguro até 200%

---

## 2. Feature Cards com Microinterações

### Implementação Avançada

```
const FeatureCard = ({
  title,
  description,
  icon,
  children
}) => {
  const [isExpanded, setIsExpanded] = useState(false);

  return (
    <div
      className={`feature-card ${isExpanded ? 'expanded' : ''}`}
      aria-expanded={isExpanded}
    >
      <div className="card-header">
        <div className="icon-container">{icon}</div>
        <h3>{title}</h3>
        <button
          aria-label={isExpanded ? 'Recolher detalhes' : 'Expandir
detalhes'}
          onClick={() => setIsExpanded(!isExpanded)}
        >
```

```
          {isExpanded ? <CollapseIcon /> : <ExpandIcon />}
        </button>
      </div>
      <p>{description}</p>

      {isExpanded && (
        <div className="card-details" aria-live="polite">
          {children}
        </div>
      )}
    </div>
  );
};

// Microinterações CSS detalhadas
.feature-card {
  transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
  transform-origin: top center;

  &:hover {
    transform: translateY(-5px) scale(1.02);
    box-shadow: 0 12px 24px rgba(139, 69, 19, 0.2);
  }

  &.expanded {
    transform: scale(1.05);
    z-index: 10;

    .card-details {
      animation: slideDown 0.4s forwards;
    }
  }
}

@keyframes slideDown {
  from { opacity: 0; max-height: 0; }
  to { opacity: 1; max-height: 500px; }
}
```

---

## 3. Interactive Comparison Table

### Implementação com Acessibilidade

```
const ComparisonTable = () => {
  return (
    <div role="table" aria-label="Comparação de planos">
      <div role="rowgroup">
        <div role="row" className="comparison-header">
          <div role="columnheader" aria-sort="none">
```

```
            Recurso
          </div>
          <div role="columnheader" aria-sort="none">
            Básico
          </div>
          <div role="columnheader" aria-sort="none">
            Profissional
          </div>
        </div>
      </div>

      <div role="rowgroup">
        {features.map((feature) => (
          <div role="row" key={feature.id}>
            <div role="cell" className="feature-name">
              {feature.name}
            </div>
            <div role="cell">{feature.basic ? "✓" : "✗"}</div>
            <div role="cell">{feature.pro ? "✓" : "✗"}</div>
          </div>
        ))}
      </div>
    </div>
  );
};
```

Padrões de Decisão Aplicados

1. **Anchoring**:
    - Destaque visual no plano recomendado
    - Posicionamento estratégico no centro da tabela
2. **Social Proof**:
    - Badge "Mais Popular" com contraste aumentado
    - Ícone de verificação para recursos mais usados
3. **Progressive Enhancement**:
    - Features organizadas por complexidade
    - Informações progressivas ao passar o mouse

---

## 4. Animated Timeline/Roadmap

Implementação com React Spring

```
import { useSpring, animated } from "react-spring";
```

```
const TimelineItem = ({ status, date, title, description, progress }) =>
{
  const progressStyle = useSpring({
    width: `${progress}%`,
    from: { width: "0%" },
    config: { tension: 120, friction: 14 },
  });

  return (
    <div className={`timeline-item ${status}`}>
      <div className="timeline-marker">
        {status === "completed" && <CheckCircle />}
        {status === "in-progress" && <Loader />}
        {status === "upcoming" && <Clock />}
      </div>

      <div className="timeline-content">
        <div className="timeline-date">{date}</div>
        <h4 className="timeline-title">{title}</h4>
        <p className="timeline-description">{description}</p>

        {status === "in-progress" && (
          <div className="progress-bar">
            <animated.div
              className="progress-fill"
              style={progressStyle}
              aria-valuenow={progress}
              aria-valuemin="0"
              aria-valuemax="100"
              role="progressbar"
            />
          </div>
        )}
      </div>
    </div>
  );
};
```

---

## 5. Componentes de Prova Social

### Implementação com Swiper

```
import { Swiper, SwiperSlide } from "swiper/react";
import { Autoplay, Navigation, Pagination } from "swiper/modules";

const TestimonialCarousel = () => {
  return (
    <Swiper
      modules={[Autoplay, Navigation, Pagination]}
```

```
      spaceBetween={30}
      slidesPerView={1}
      autoplay={{ delay: 8000, disableOnInteraction: false }}
      navigation
      pagination={{ clickable: true }}
      aria-label="Depoimentos de clientes"
    >
      {testimonials.map((testimonial) => (
        <SwiperSlide key={testimonial.id}>
          <TestimonialCard {...testimonial} />
        </SwiperSlide>
      ))}
    </Swiper>
  );
};

const TestimonialCard = ({ avatar, quote, author, role, rating }) => (
  <blockquote className="testimonial">
    <div className="testimonial-content">
      <img
        src={avatar}
        alt={`Foto de ${author}`}
        className="testimonial-avatar"
      />
      <p className="testimonial-quote">"{quote}"</p>
    </div>
    <footer className="testimonial-footer">
      <cite className="testimonial-author">{author}</cite>
      <span className="testimonial-role">{role}</span>
      <div className="testimonial-rating">
        {[...Array(5)].map((_, i) => (
          <StarIcon key={i} filled={i < rating} />
        ))}
      </div>
    </footer>
  </blockquote>
);
```

---

## 6. Interactive Demo Components

### Implementação com Framer Motion

```
import { motion, AnimatePresence } from "framer-motion";

const InteractiveDemo = ({ scenario }) => {
  const [step, setStep] = useState(0);

  const steps = {
    "weather-alert": [
```

```
      { action: "receive-notification", title: "Receber alerta" },
      { action: "open-app", title: "Abrir aplicativo" },
      { action: "view-details", title: "Ver detalhes" },
      { action: "take-action", title: "Tomar ação" },
    ],
  };

  const currentStep = steps[scenario][step];

  return (
    <div className="interactive-demo">
      <div className="demo-progress">
        {steps[scenario].map((s, i) => (
          <button
            key={i}
            className={`step-indicator ${i === step ? "active" : ""}`}
            onClick={() => setStep(i)}
            aria-label={`Passo ${i + 1}: ${s.title}`}
          />
        ))}
      </div>

      <AnimatePresence mode="wait">
        <motion.div
          key={step}
          initial={{ opacity: 0, x: 20 }}
          animate={{ opacity: 1, x: 0 }}
          exit={{ opacity: 0, x: -20 }}
          transition={{ duration: 0.3 }}
          className="demo-content"
        >
          <DemoStep action={currentStep.action} />
        </motion.div>
      </AnimatePresence>
    </div>
  );
};
```

---

## 7. Accessibility-First Components

### Implementação de Alto Contraste

```
const AccessibilityToggle = () => {
  const [highContrast, setHighContrast] = useState(false);

  useEffect(() => {
    if (highContrast) {
      document.documentElement.setAttribute('data-contrast', 'high');
    } else {
```

```
      document.documentElement.removeAttribute('data-contrast');
    }
  }, [highContrast]);

  return (
    <div className="accessibility-toolbar">
      <ToggleButton
        pressed={highContrast}
        onClick={() => setHighContrast(!highContrast)}
        aria-pressed={highContrast}
      >
        <HighContrastIcon />
        <span>Alto Contraste</span>
      </ToggleButton>

      <FontSizeControl />
      <ReadingModeToggle />
    </div>
  );
};

// CSS para modo alto contraste
[data-contrast="high"] {
  --text-color: #000;
  --bg-color: #fff;
  --primary: #0057B7;
  --secondary: #FFD700;

  * {
    color: var(--text-color) !important;
    background-color: var(--bg-color) !important;
  }

  a, button {
    border: 2px solid var(--text-color) !important;
  }
}
```

---

## 8. Smart CTA Sections

Implementação com Context API

```
const PersonaCTAs = () => {
  const { persona } = usePersonaContext();

  return (
    <section className="cta-section">
      {persona === "small" && (
        <div className="cta-small">
```

```
          <WhatsAppCTA />
          <SupportContact />
        </div>
      )}

      {persona === "medium" && (
        <div className="cta-medium">
          <FreeTrialButton />
          <DemoSchedule />
        </div>
      )}

      {persona === "innovative" && (
        <div className="cta-innovative">
          <APIDocsLink />
          <GitHubRepository />
        </div>
      )}
    </section>
  );
};

const WhatsAppCTA = () => (
  <a
    href="https://wa.me/5511999999999"
    className="whatsapp-cta"
    aria-label="Falar com especialista via WhatsApp"
  >
    <WhatsAppIcon />
    <span>Falar com Especialista</span>
  </a>
);
```

---

## 9. Feature Discovery Patterns

### Implementação com Gamificação

```
const OnboardingProgress = () => {
  const [progress, setProgress] = useState([
    { id: 1, completed: true, title: "Criar conta" },
    { id: 2, completed: false, title: "Configurar fazenda" },
    { id: 3, completed: false, title: "Primeiro alerta" },
  ]);

  const completeStep = (id) => {
    setProgress((prev) =>
      prev.map((step) => (step.id === id ? { ...step, completed: true }
: step))
    );
```

```
  };

  return (
    <div className="onboarding-tracker">
      <h3>Seu progresso</h3>
      <div className="steps">
        {progress.map((step) => (
          <div
            key={step.id}
            className={`step ${step.completed ? "completed" : ""}`}
            aria-current={
              !step.completed &&
              !progress.find((s) => s.id < step.id && !s.completed)
            }
          >
            <div className="step-icon">
              {step.completed ? <CheckIcon /> : <StepNumber number=
{step.id} />}
            </div>
            <span className="step-title">{step.title}</span>
          </div>
        ))}
      </div>

      <Achievements unlocked={progress.filter((s) =>
s.completed).length} />
    </div>
  );
};
```

---

## 10. Performance Optimization Patterns

### Implementação de Lazy Loading

```
const LazyFeatureLoader = () => {
  return (
    <Suspense fallback={<FeatureSkeleton />}>
      <LazyComponent />
    </Suspense>
  );
};

const LazyComponent = lazy(() => import("./HeavyFeatureComponent"));

const FeatureSkeleton = () => (
  <div className="skeleton-loader">
    <div className="skeleton-header"></div>
    <div className="skeleton-content"></div>
    <div className="skeleton-footer"></div>
```

```
  </div>
);

// Otimização de imagens com Next.js
<Image
  src="/coffee-field.jpg"
  alt="Plantação de café"
  width={800}
  height={450}
  placeholder="blur"
  blurDataURL="..."
  quality={85}
  sizes="(max-width: 768px) 100vw, 50vw"
/>;
```

---

## 11. Motion Design Guidelines

Implementação com Princípios de Animação

```
/* Animação de entrada suave */
@keyframes fadeIn {
  from {
    opacity: 0;
    transform: translateY(20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

/* Animação de destaque */
@keyframes pulse {
  0% {
    transform: scale(1);
  }
  50% {
    transform: scale(1.05);
  }
  100% {
    transform: scale(1);
  }
}

/* Transições de microinterações */
.button {
  transition: transform 0.2s ease-out, box-shadow 0.2s ease-out,
    background-color 0.3s ease;
```

```
  &:hover {
    transform: translateY(-2px);
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  }

  &:active {
    transform: translateY(1px);
  }
}

/* Animações de feedback */
.form-input {
  transition: border-color 0.3s ease;

  &:focus {
    border-color: #0070f3;
    box-shadow: 0 0 0 3px rgba(0, 118, 255, 0.2);
  }

  &.invalid {
    animation: shake 0.5s;
    border-color: #e53e3e;
  }
}

@keyframes shake {
  0%,
  100% {
    transform: translateX(0);
  }
  20%,
  60% {
    transform: translateX(-5px);
  }
  40%,
  80% {
    transform: translateX(5px);
  }
}
```

# Considerações Finais

Este documento detalhado fornece implementações práticas e acessíveis para cada componente, com foco em:

- Experiência do usuário fluída e intuitiva
- Performance otimizada para diferentes dispositivos
- Acessibilidade seguindo diretrizes WCAG 2.1
- Manutenção e escalabilidade do código

- Alinhamento com as personas do projeto Global Coffee

Os componentes podem ser implementados progressivamente conforme a necessidade do projeto.