IntelliThesis Tech Stack Interview Questions & Answers

Table of Contents

Frontend (Next.js/React/TypeScript)

Next.js 15

Q1: What's the difference between the Pages Router and App Router in Next.js?

Answer:

- **Pages Router**: File-based routing where each file in pages/ becomes a route. Uses getStaticProps, getServerSideProps for data fetching.
- **App Router**: Directory-based routing using app/ folder with layout.tsx, page.tsx, loading.tsx files. Uses Server Components by default and has built-in data fetching patterns.

Q2: How would you implement server-side rendering for dynamic content in Next.js 15?

Answer:

```
// In App Router
async function Page({ params }: { params: { id: string } }) {
  const data = await fetch(`/api/posts/${params.id}`);
  const post = await data.json();

  return <div>{post.title}</div>;
}
```

Q3: Explain the benefits of using Next.js Image component over regular img tags.

Answer:

- Automatic image optimization and compression
- Lazy loading by default
- Responsive images with multiple sizes
- WebP format conversion when supported
- Prevents layout shift with proper sizing

React 19

Q4: What are the new features in React 19 and how would you use them?

Answer:

- **Actions**: Server functions that can be called from client components
- **use() hook**: For consuming promises and context
- **Document Metadata**: Built-in support for managing document head
- **Improved hydration**: Better error handling and performance

Q5: Explain the difference between useState and useReducer, and when would you choose one over the other?

Answer:

- **useState**: For simple state management with primitive values
- **useReducer**: For complex state logic with multiple sub-values or when next state depends on previous state
- Choose useReducer when state logic is complex or involves multiple related state updates

TypeScript

Q6: How would you type a function that accepts different types of parameters?

Answer:

```
// Union types
function processData(data: string | number | boolean) {
  // implementation
}

// Generic function
function identity<T>(arg: T): T {
  return arg;
}

// Function overloads
function combine(a: string, b: string): string;
function combine(a: number, b: number): number;
function combine(a: any, b: any): any {
  return a + b;
}
```

Tailwind CSS

Q7: How would you create a responsive design using Tailwind CSS?

Answer:

```
<div class="w-full md:w-1/2 lg:w-1/3 xl:w-1/4">
  <div class="text-sm md:text-base lg:text-lg">
    Responsive content
  </div>
</div>
```

Backend (Express.js/Node.js)

Express.js

Q8: How do you implement middleware in Express.js and what's the order of execution?

Answer:

```
// Middleware order matters - executed in sequence
app.use(express.json()); // Parse JSON bodies
app.use(cors()); // Handle CORS
app.use(helmet()); // Security headers
app.use('/api', apiRoutes); // Route-specific middleware
```

Q9: How would you handle file uploads in Express.js?

Answer:

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });

app.post('/upload', upload.single('file'), (req, res) => {
  const file = req.file;
  // Process uploaded file
});
```

Authentication & Security

Q10: How does JWT authentication work and what are its advantages/disadvantages?

Answer:

- **How it works**: Server creates a signed token containing user data, client stores and sends token with requests
- **Advantages**: Stateless, scalable, works across domains
- **Disadvantages**: Can't be revoked before expiration, size limitations, security depends on proper implementation

Q11: Explain the difference between bcrypt and other hashing algorithms.

Answer:

- **bcrypt**: Adaptive algorithm with salt rounds, slow by design to prevent brute force attacks
- **MD5/SHA1**: Fast but vulnerable to rainbow table attacks
- **bcrypt** is preferred for password hashing due to its security features

---

Python Backend (FastAPI)

FastAPI

Q12: What are the advantages of FastAPI over Flask or Django?

Answer:

- **Performance**: Built on Starlette and Pydantic, very fast
- **Type hints**: Full Python type annotation support

- **Auto documentation**: Automatic OpenAPI/Swagger docs
- **Modern async**: Built-in async/await support
- **Validation**: Automatic request/response validation with Pydantic

Q13: How do you implement dependency injection in FastAPI?

Answer:

```
from fastapi import Depends

def get_db():
    db = Database()
    try:
        yield db
    finally:
        db.close()

@app.get("/users/")
def read_users(db: Database = Depends(get_db)):
    return db.get_users()
```

AI/ML Integration

Q14: How would you integrate OpenAI's GPT-4 API into a FastAPI application?

Answer:

```
from openai import OpenAI
from fastapi import FastAPI

client = OpenAI(api_key="your-api-key")

@app.post("/analyze")
async def analyze_document(content: str):
    response = client.chat.completions.create(
        model="gpt-4",
        messages=[{"role": "user", "content": f"Analyze: {content}"}]
    )
    return {"analysis": response.choices[0].message.content}
```

DevOps & Deployment

Docker & Containerization

Q15: How would you containerize a multi-service application like IntelliThesis?

Answer:

```
# Frontend Dockerfile
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
EXPOSE 3000
CMD ["npm", "start"]
```

Q16: What are the best practices for writing Dockerfiles?

Answer:

- Use multi-stage builds to reduce image size
- Copy package files first to leverage Docker layer caching
- Use specific base image versions
- Run as non-root user for security
- Minimize the number of layers

CI/CD

Q17: How would you set up automated testing in a CI/CD pipeline?

Answer:

```
# GitHub Actions example
name: CI/CD Pipeline
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Install dependencies
        run: npm install
      - name: Run tests
        run: npm test
      - name: Build
        run: npm run build
```

System Design & Architecture

Microservices

Q18: How would you design the communication between the three services in IntelliThesis?

Answer:

- **Frontend ↔ Express Server**: REST API for authentication and user management
- **Express Server ↔ FastAPI**: REST API for AI processing requests
- **Frontend ↔ FastAPI**: Direct API calls for real-time features

- Use message queues (Redis/RabbitMQ) for async processing

Q19: What are the challenges of maintaining consistency across microservices?

Answer:

- **Data consistency**: Implementing distributed transactions or eventual consistency
- **Service discovery**: Managing service endpoints and load balancing
- **Monitoring**: Distributed tracing and logging across services
- **Deployment coordination**: Ensuring compatible versions across services

Performance & Scalability

Q20: How would you optimize the performance of the AI document analysis feature?

Answer:

- **Caching**: Redis cache for processed documents
- **Async processing**: Queue-based processing for large documents
- **CDN**: Serve static assets globally
- **Database optimization**: Proper indexing and query optimization
- **Load balancing**: Distribute AI processing across multiple instances

---

Advanced Topics

Real-time Features

Q21: How would you implement real-time chat functionality?

Answer:

```
// Using Socket.IO
const io = require('socket.io')(server);

io.on('connection', (socket) => {
  socket.on('join-room', (roomId) => {
    socket.join(roomId);
  });

  socket.on('send-message', (data) => {
    io.to(data.roomId).emit('new-message', data);
  });
});
```

Testing

Q22: How would you write unit tests for React components?

Answer:

```
import { render, screen, fireEvent } from '@testing-library/react';
import { CodeEditor } from './CodeEditor';

test('renders code editor with default value', () => {
  render(<CodeEditor defaultValue="console.log('hello')" />);
  expect(screen.getByText("console.log('hello')")).toBeInTheDocument();
});
```

Conclusion

This guide covers the essential technical knowledge required for working with the IntelliThesis tech stack. The questions range from basic concepts to advanced implementation details, providing a comprehensive assessment of a candidate's skills.

Key Technologies Covered:

- Next.js 15, React 19, TypeScript
- Express.js, Node.js, MongoDB/PostgreSQL
- FastAPI, Python, AI/ML integration
- Docker, CI/CD, DevOps
- System design and architecture

Preparation Tips:

1. Practice implementing these concepts in real projects
2. Understand the trade-offs between different approaches
3. Stay updated with the latest versions and best practices
4. Focus on practical implementation rather than just theoretical knowledge