Parsing JWT Headers Across Programming Paradigms

Aidan Pace

[2025-04-28 Mon]

Introduction

Parsing JWT Headers Across Programming Paradigms

S1cmss0• A cross-language exploration of JWT header parsing techniques

What We'll Cover

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025

What We'll Cover

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution
- JWT structure and fundamentals (beginner-friendly)

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution
- JWT structure and fundamentals (beginner-friendly)
- Base64url encoding challenges

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution
- JWT structure and fundamentals (beginner-friendly)
- Base64url encoding challenges
- Header parsing patterns across languages

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution
- JWT structure and fundamentals (beginner-friendly)
- Base64url encoding challenges
- Header parsing patterns across languages
- Functional vs object-oriented approaches

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution
- JWT structure and fundamentals (beginner-friendly)
- Base64url encoding challenges
- Header parsing patterns across languages
- Functional vs object-oriented approaches
- Language-specific idioms and best practices

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution
- JWT structure and fundamentals (beginner-friendly)
- Base64url encoding challenges
- Header parsing patterns across languages
- Functional vs object-oriented approaches
- Language-specific idioms and best practices
- Security considerations and common attacks

Introduction

Parsing JWT Headers Across Programming Paradigms

- A cross-language exploration of JWT header parsing techniques
- SPLASH/StrangeLoop/PyConf/RacketCon/EuroLISP 2025
- Aidan Pace (@aygp-dr)

What We'll Cover

- Historical context and authentication evolution
- JWT structure and fundamentals (beginner-friendly)
- Base64url encoding challenges
- Header parsing patterns across languages
- Functional vs object-oriented approaches
- Language-specific idioms and best practices
- Security considerations and common attacks

Authentication Evolution

Historical Context of Authentication

- Early authentication: Username/password pairs

Authentication Evolution

Historical Context of Authentication

- Early authentication: Username/password pairs
- Server-side sessions with cookies (stateful)

Authentication Evolution

Historical Context of Authentication

- Early authentication: Username/password pairs
- Server-side sessions with cookies (stateful)
- Token-based authentication emergence (stateless)

Authentication Evolution

Historical Context of Authentication

- Early authentication: Username/password pairs
- Server-side sessions with cookies (stateful)
- Token-based authentication emergence (stateless)
- JWT standardization (RFC 7519, May 2015)

Authentication Evolution

Historical Context of Authentication

- Early authentication: Username/password pairs
- Server-side sessions with cookies (stateful)
- Token-based authentication emergence (stateless)
- JWT standardization (RFC 7519, May 2015)
- Modern authentication flows (OAuth 2.0, OIDC)

JWT Basics

JWT Structure Refresher

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIn0.dozjgNryP4J
```

Three dot-separated base64url-encoded segments:

1. Header (algorithm & token type)
2. Payload (claims)
3. Signature

```
digraph {
  rankdir=LR;
  node [shape=box, style=filled, fillcolor="#e6f3ff", fontname="monospace"];
  edge [fontname="Arial"];
```

Language Implementations

JavaScript (Browser)

```
const authHeader = "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOi...
const token = authHeader.split(' ')[1];

// IMPORTANT: In production, verify signature before parsing!
// This example is for demonstration only

// Decode the header part
const headerPart = token.split('.')[0];
const decodedHeader = JSON.parse(atob(headerPart));
console.log(decodedHeader);
```

Note: `atob()` handles base64 but not base64url specifically

Analysis

Common Patterns & Variations

1. Token extraction: Split by space or regex

Cross-Language Performance Analysis

Language	Parsing Time (s)	Memory Usage (KB)
Rust	5.2	1.8
JavaScript	24.7	12.3

Analysis

Common Patterns & Variations

1. Token extraction: Split by space or regex
2. Base64url handling:

Cross-Language Performance Analysis

Language	Parsing Time (s)	Memory Usage (KB)
Rust	5.2	1.8
JavaScript	24.7	12.3

Analysis

Common Patterns & Variations

1. Token extraction: Split by space or regex
2. Base64url handling:
   - Character replacement (- → +, _ → /)

Cross-Language Performance Analysis

Language	Parsing Time (s)	Memory Usage (KB)
Rust	5.2	1.8
JavaScript	24.7	12.3

Analysis

Common Patterns & Variations

1. Token extraction: Split by space or regex
2. Base64url handling:
   - Character replacement (- → +, _ → /)
   - Padding calculation

Cross-Language Performance Analysis

Language	Parsing Time (s)	Memory Usage (KB)
Rust	5.2	1.8
JavaScript	24.7	12.3

Analysis

Common Patterns & Variations

1. Token extraction: Split by space or regex
2. Base64url handling:
   - Character replacement (- → +, _ → /)
   - Padding calculation
   - URL-safe decoder availability (JVM advantage)

Cross-Language Performance Analysis

Language	Parsing Time (s)	Memory Usage (KB)
Rust	5.2	1.8
JavaScript	24.7	12.3

Analysis

Common Patterns & Variations

1. Token extraction: Split by space or regex
2. Base64url handling:
   - Character replacement (- → +, _ → /)
   - Padding calculation
   - URL-safe decoder availability (JVM advantage)
3. JSON parsing: Native vs libraries

Cross-Language Performance Analysis

Language	Parsing Time (s)	Memory Usage (KB)
Rust	5.2	1.8
JavaScript	24.7	12.3

Analysis

Common Patterns & Variations

1. Token extraction: Split by space or regex
2. Base64url handling:
   - Character replacement (- → +, _ → /)
   - Padding calculation
   - URL-safe decoder availability (JVM advantage)
3. JSON parsing: Native vs libraries
4. Error handling: Idiomatic differences

Cross-Language Performance Analysis

Language	Parsing Time (s)	Memory Usage (KB)
Rust	5.2	1.8
JavaScript	24.7	12.3

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload

Common JWT Attacks

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)

Common JWT Attacks

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)

Common JWT Attacks

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)

Common JWT Attacks

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)
- Token tampering - Modifying claims without invalidating signature

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)
- Token tampering - Modifying claims without invalidating signature
- Token injection - Using a token from one context in another

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)
- Token tampering - Modifying claims without invalidating signature
- Token injection - Using a token from one context in another
- Replay attacks - Reusing captured tokens

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)
- Token tampering - Modifying claims without invalidating signature
- Token injection - Using a token from one context in another
- Replay attacks - Reusing captured tokens

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)
- Token tampering - Modifying claims without invalidating signature
- Token injection - Using a token from one context in another
- Replay attacks - Reusing captured tokens

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)
- Token tampering - Modifying claims without invalidating signature
- Token injection - Using a token from one context in another
- Replay attacks - Reusing captured tokens

Security Considerations

JWT Security Best Practices

- Always verify signatures before parsing or using payload
- Use strong algorithms (prefer RS256/ES256 over HS256)
- Implement proper key management (rotation, secure storage)
- Set appropriate token lifetimes (short-lived access tokens)
- Include essential claims (iss, sub, exp, aud, iat)

Common JWT Attacks

- "alg": "none" attack - Attacker removes signature validation requirement
- Algorithm confusion - Switching from asymmetric (RS256) to symmetric (HS256)
- Token tampering - Modifying claims without invalidating signature
- Token injection - Using a token from one context in another
- Replay attacks - Reusing captured tokens

Real-World Applications

Cross-Language Implementation Comparison

Feature	JavaScript	Python	Rust	Clojure	TypeScript
Type Safety	Limited	Optional	Strong	Dynamic	Strong
Base64 Handling	Manual	Built-in	Crates	JVM	Manual
Error Handling	try/catch	Exceptions	Result	Monadic	try/catch
Performance	Medium	Low	High	Medium	Medium
JWT Libraries	Many	Several	Few	Few	Many

JWT in Production

- API Gateway token validation

Real-World Applications

Cross-Language Implementation Comparison

Feature	JavaScript	Python	Rust	Clojure	TypeScript
Type Safety	Limited	Optional	Strong	Dynamic	Strong
Base64 Handling	Manual	Built-in	Crates	JVM	Manual
Error Handling	try/catch	Exceptions	Result	Monadic	try/catch
Performance	Medium	Low	High	Medium	Medium
JWT Libraries	Many	Several	Few	Few	Many

JWT in Production

- API Gateway token validation
- Microservice authorization

Real-World Applications

Cross-Language Implementation Comparison

Feature	JavaScript	Python	Rust	Clojure	TypeScript
Type Safety	Limited	Optional	Strong	Dynamic	Strong
Base64 Handling	Manual	Built-in	Crates	JVM	Manual
Error Handling	try/catch	Exceptions	Result	Monadic	try/catch
Performance	Medium	Low	High	Medium	Medium
JWT Libraries	Many	Several	Few	Few	Many

JWT in Production

- API Gateway token validation
- Microservice authorization
- Single Sign-On implementations

Real-World Applications

Cross-Language Implementation Comparison

Feature	JavaScript	Python	Rust	Clojure	TypeScript
Type Safety	Limited	Optional	Strong	Dynamic	Strong
Base64 Handling	Manual	Built-in	Crates	JVM	Manual
Error Handling	try/catch	Exceptions	Result	Monadic	try/catch
Performance	Medium	Low	High	Medium	Medium
JWT Libraries	Many	Several	Few	Few	Many

JWT in Production

- API Gateway token validation
- Microservice authorization
- Single Sign-On implementations
- Mobile app authentication

Real-World Applications

Cross-Language Implementation Comparison

Feature	JavaScript	Python	Rust	Clojure	TypeScript
Type Safety	Limited	Optional	Strong	Dynamic	Strong
Base64 Handling	Manual	Built-in	Crates	JVM	Manual
Error Handling	try/catch	Exceptions	Result	Monadic	try/catch
Performance	Medium	Low	High	Medium	Medium
JWT Libraries	Many	Several	Few	Few	Many

JWT in Production

- API Gateway token validation
- Microservice authorization
- Single Sign-On implementations

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency

Debugging Tools

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization

Debugging Tools

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization
- Malformed tokens - Inspect encoding, ensure proper base64url handling

Debugging Tools

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization
- Malformed tokens - Inspect encoding, ensure proper base64url handling
- Missing claims - Validate token structure against expected schema

Debugging Tools

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization
- Malformed tokens - Inspect encoding, ensure proper base64url handling
- Missing claims - Validate token structure against expected schema
- Algorithm mismatch - Confirm header alg matches implementation

Debugging Tools

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization
- Malformed tokens - Inspect encoding, ensure proper base64url handling
- Missing claims - Validate token structure against expected schema
- Algorithm mismatch - Confirm header alg matches implementation

Debugging Tools

- Online JWT debugger (jwt.io)

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization
- Malformed tokens - Inspect encoding, ensure proper base64url handling
- Missing claims - Validate token structure against expected schema
- Algorithm mismatch - Confirm header alg matches implementation

Debugging Tools

- Online JWT debugger (jwt.io)
- Language-specific JWT libraries with debug options

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization
- Malformed tokens - Inspect encoding, ensure proper base64url handling
- Missing claims - Validate token structure against expected schema
- Algorithm mismatch - Confirm header alg matches implementation

Debugging Tools

- Online JWT debugger (jwt.io)
- Language-specific JWT libraries with debug options
- Base64 inspection tools

Debugging & Troubleshooting

Common JWT Issues and Solutions

- Invalid signature - Check key matching, algorithm consistency
- Expired tokens - Verify client/server clock synchronization
- Malformed tokens - Inspect encoding, ensure proper base64url handling
- Missing claims - Validate token structure against expected schema
- Algorithm mismatch - Confirm header alg matches implementation

Debugging Tools

- Online JWT debugger (jwt.io)
- Language-specific JWT libraries with debug options
- Base64 inspection tools
- Request/response inspection with developer tools

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing

Conclusion

Cross-Paradigm Insights

Paradigm	Strengths	JWT Application
Object-Oriented	Encapsulation, inheritance	Token with validation methods
Functional	Composition, immutability	Transform pipeline for parsing
Procedural	Simplicity, performance	Lightweight validators
Reactive	Event handling	Token verification in async flows

Takeaways

1. Base64url encoding requires special attention
2. Each language has idiomatic parsing advantages
3. Functional approaches shine for transformation pipelines
4. Security first: always verify signatures before parsing