Aeria

João G. Santos

github.com/minenwerfer

O que é o Aeria?

O Aeria é um ecossistema para a criação de aplicações com padrão industrial com ênfase especial em segurança. O Aeria pode ser usado como backend standalone, como aplicação fullstack, ou usado em conjunto com outras tecnologias, a exemplo do NextJS.

As seguintes partes compõe o ecossistema:

- **Aeria Core**: toolkit para criar backends seguros com tipagem ponta-a-ponta e reflexão
- **Aeria Lang**: linguagem com primitivas para criar entidades do banco de dados e testes end-to-end (semelhante ao Prisma)
- **Aeria UI**, **Aeria SDK**, pacotes de idioma, language servers, plugins de IDE, plugins da comunidade, etc

Com exceção do Aeria Lang, todos os componentes encontram-se prontos para a produção.

Filosofia

O Aeria tem como objetivo fornecer primitivas de segurança da informação e de acesso a dados por meio de uma linguagem que transpila para JavaScript e TypeScript.

Um programador consegue prototipar uma API usando um único arquivo `.aeria` e depois estendê-la usando JavaScript ou TypeScript. O runtime Aeria por sua vez executa a API observando as garantias de segurança.

```
collection Transaction {
  immutable true

  properties {
    sender Person
    receiver Person
    amount number @exclusiveMinimum(0)
  }

  functions {
    get
    getAll
    send
  }

  security {
    send {
      rateLimiting {
        strategy tenant
        scale 5
      }
    }
  }
}
```

Uma feature da API pode ser implementada em poucas linhas. A superfície do código de uma aplicação Aeria é reduzida numa escala de dezenas de vezes.

Conceito

- o arquivo de entrypoint (`index.js`) exporta os metadados da API estaticamente, sendo esses compostos por:
  - collections
  - routes
- durante o tempo de execução esses metadados são obtidos através de *imports dinâmicos* cujo cache é gerenciado pelo NodeJS
- como os metadados estão disponíveis estaticamente, através de um `typeof import('.')` é possível obter o tipo de qualquer schema e endpoint
  - avaliação preguiçosa dos tipos é usada para evitar erros de circularidade, uma vez que os tipos obtidos dessa forma são fundalmentalmente circulares

À época dos estágios iniciais de desenvolvimento do Aeria, muitas das features escolhidas só eram possíveis nos nightly releases do TypeScript e NodeJS. O Aeria é o estado da arte dessas duas ferramentas.

O que significa "estaticamente"?

Significa que qualquer ferramenta capaz de importar um arquivo JavaScript consegue fazer o parse dos metadados sem executar a API. Isso permite expor a API a geradores de documentação, geradores de relatório de segurança, etc.

```
$ node -e "console.log(require('./dist/index.js').collections)"
{
  user: {
    description: {
      ...
    },
    security: {
      authenticate: {
        rateLimiting: {
          strategy: 'ip',
          scale: 100
        }
      }
    }
  }
  ...
}
```

Segurança nativa

A distribuição padrão do Aeria contém as seguintes funcionalidades já implementadas:

- Validação de dados em runtime
- Rate limiting por duas estratégias (tenant e IP)
- Autenticação e controle de acesso baseado em regras (RBAC)
- Ownership e imutabilidade
- Log rico de ações efetuadas por usuários

Falhas comuns relacionadas à implementação do controle de acesso, tokens, IDOR, MAID, ausência de rate limiting, NoSQL injection, etc, são coibidas até mesmo por desenvolvedores com pouca familiaridade com App Sec.

Exemplo de rate limiting

O Aeria oferece duas formas para fazer garantias de segurança: uma forma declarativa, e uma forma funcional, com a qual o desenvolvedor possui mais liberdade.

```
{
  security: {
    rateLimiting: {
      insert: {
        strategy: 'ip',
        scale: 5
      }
    }
  }
}
```

Forma declarativa: a estratégia e a frequência do rate limiting da função insert são definidos por um objeto

```
router.GET('/resource', async (context) => {
  const rateEither = await context.limitRate({
    strategy: 'ip',
    scale: 5
  })

  if( isLeft(rateEither) ) {
    return rateEither
  }

  // lógica da função
})
```

Forma funcional: o programador possui mais versatilidade e pode por exemplo notificar o usuário por email quando o ceiling é atingido

Accountability

Imagine o seguinte lugar-comum de sistemas de gestão:

Uma empresa possui um sistema para controle de logística ao qual vários usuários de diferentes setores possuem acesso. Nesse sistema existe uma ação manual para importação de pedidos de marketplaces por data, para que esses possam ser expedidos aos respectivos clientes.

Um usuário, por incautela, importa pedidos de uma faixa de datas indevida, o setor de separação de pedidos não é avisado a tempo, e isso acarreta prejuízos para a empresa. Agora, para que o caso seja investigado, deve-se conhecer as seguintes informações:

- o usuário que performou a ação
- a data da ação
- qual a origem dos pedidos que foram importados

Logs do servidor HTTP não são ricos o suficiente para serem úteis e, caso uma função que faz o log das informações supracitadas não tiver sido invocada, as mesmas foram perdidas para sempre.

Logging

O Aeria oferece a garantia de que qualquer ação crítica executada no sistema poderá ser posteriormente auditada, o que oferece uma gigantesca vantagem no gerenciamento de incidentes. Essa é uma das grandes propostas da ferramenta: sistemas auditáveis por design.

Forma declarativa

```
{
  functions: {
    importOrders
  },
  security: {
    importOrders: {
      logging: {
        strategy: 'tenant',
        level: 'critical',
      }
    }
  }
}
```

Invocação manual da função de logging

```
router.POST('/importOrders', async (context) => {
  await context.log({
    strategy: 'tenant',
    level: 'critical',
    // contexto adicional...
  })

  // lógica da função
})
```

Verificações a cada push

O Aeria Lang inclui um DSL para testes end-to-end. Ao executar o comando `aeria test` obtém-se um sumário com os testes bem e mal sucedidos.

Com poucas linhas de código é possível testar o controle de acesso, formato do payload, rate limiting, etc. É trivial incluir os testes numa etapa do CI para que sejam executados a cada push.

```
test POST /person/insert {
  token {
    roles []
  }
  payload {
    what {
      name "joao"
      age 50
    }
  }
  response left {
    error "AUTHORIZATION_ERROR"
  }
}
```

No exemplo acima, é asserta-se que ao tentar fazer um 'POST /person/insert' com um usuário não-autenticado, a resposta deve ser um Left com o erro 'AUTHORIZATION_ERROR'.

Integração com vendors

O Aeria oferece built-ins para autenticação, gerenciamento de arquivos, etc, no entanto pode ser que eles sejam insuficientes para aplicações que dependem criticamente dessas funcionalidades.

Provedores de serviços como buckets de armazenamento, mailing, servidores de identidade, etc, podem ser usados no lugar dos equivalentes built-ins por meio de uma API de drivers.

```
import s3 from '@aeria-plugins/s3'
import config from './config.js'

export default init({
  config,
  drivers: {
    files: s3({
      key: process.env.S3_SECRET_KEY
    })
  },
})
```

No exemplo acima, é asserta-se que ao tentar fazer um 'POST /person/insert' com um usuário não-autenticado, a resposta deve ser um Left com o erro 'AUTHORIZATION_ERROR'.

Resumo

A intenção desse slide foi demonstrar que com o Aeria é possível criar um modelo de software on demand que muito dificilmente seria possível com métodos e tecnologias tradicionais.

Com uma divisão de trabalho simples e uma folha salarial bem reduzida, uma equipe seria capaz de desenvolver sistemas seguros para o enterprise, suprindo um gap que existe hoje no mercado tecnológico, sobretudo o brasileiro, que é caro, pouco eficiente, e imaturo com relação a App Sec.

Fim

Obrigado pela atenção!