PUC Minas
Virtual

Arquitetura de Front-Ends

Síntese da disciplina

Samuel Martins

PUC Minas
Virtual

Sumário

UNIDADE I – INTRODUÇÃO À WEB E FRAMEWORKS DE DESENVOLVIMENTO

PERFIL PROFISSIONAL

O perfil do profissional de Arquitetura de front-end vem sendo requisitado nas organizações devido a alta demanda de aplicações web cada vez mais complexas. Esse tipo de profissional atua em conjunto com várias especialidades, sempre traduzindo demandas de negócio em projetos consistentes. Podemos citar três camadas de atuação de um profissional com essa especialização: Contexto, Sistema e Componente.

CONTEXTO

A pessoa sempre irá atuar em alguma companhia que possui suas demandas de negócio específicas e a sua própria cultura (incluindo ambiente, budget, mercado alvo e etc). Um profissional que atua com tecnologia no mercado financeiro possui características técnicas e demandas diferentes de quem trabalha com algum produto voltado para marketing digital. O primeiro irá atuar muito mais focado em segurança, sendo que o segundo deverá se atentar para atributos que melhoram o seu produto e de seus clientes nos resultados de busca.

SISTEMA

Um profissional de Arquitetura deve atuar em diversas partes do processo de desenvolvimento. Na camada "Sistema" temos os seguintes tópicos:

- Abordagens arquiteturais:
  - Multi Page | Single Page Application (SPA) | Progressive Web Apps (PWA) | Serverless Computing | Micro frontends.
- Ambiente de desenvolvimento:
  - Frameworks | Gerenciadores de dependências | Empacotadores (bundlers).
- Processo de desenvolvimento:
  - Testes | CI/CD | Cultura DevOps.

COMPONENTE

Parte mais próxima do desenvolvimento e codificação, constituída por:

- Marcação HTML:
  - HTML semântico | Acessibilidade | Performance.
- CSS:
  - Style guides | Padrão de código (OOCSS, BEM, CSS Funcional) | Pré-processadores | Responsividade.
- Linguagem JavaScript:
  - Recursos da linguagem | Transpiladores | Especificações (ES7).
- Segurança e Infraestrutura:
  - Open Authorization (OAuth) | Cross-origin (CORS) | Cross-site Scripting (XSS).
- Otimização no Front End:
  - Avaliação de desempenho | Redução de requisições | Otimização de conteúdo | Controle de Cache.

RESPONSABILIDADES

Dentre as principais responsabilidades e skills de um profissional de arquitetura, podemos destacar o **desapego a quaisquer frameworks específicos** (os frameworks serão utilizados e abordados como ferramentas para atingir um objetivo maior) e **capacidade de fazer provas de conceito** ou POCs (*Proof of Concepts*), uma vez que em vários cenários de incertezas iremos nos deparar com projetos que demandam uma stack ainda desconhecida. Outras responsabilidades como **direcionar a visão técnica dos produtos web**, **oferecer consultoria ao time de desenvolvimento** e **ser o(a) guardiã(o) dos componentes compartilhados entre os times** também fazem parte das responsabilidades do papel.

Em se tratando de soft skills, algumas características são bastante demandadas pelo mercado, dentre elas: **disseminação de conhecimento**, uma vez que um profissional que chega ao cargo de arquitetura, normalmente já transitou em algumas outras áreas (principalmente a de desenvolvimento). Isso faz com que esse tipo de profissional tenha muito a compartilhar com os demais colegas. Isso pode ser feito tanto através de treinamentos ou mesmo no dia a dia através da análise de pull requests (seguindo a estrutura do GitFlow). **Ser liderança e referência técnica para a equipe e clientes**, uma vez que esse papel estará em constante contato com a ponta da entrega do projeto.

AMBIENTE DE ATUAÇÃO E DESENVOLVIMENTO

O ambiente de desenvolvimento front-end pode ser separado em quatro grandes camadas: world wide web (protocolo http, websocket...), browser engines, browser, tecnologia e abordagens arquiteturais.

PROTOCOLO HTTP

O protocolo utiliza o padrão requisição/resposta para comunicação entre cliente e servidor. Nesse protocolo, temos o que chamamos de verbos HTTP, que são as ações nomeadas para cada finalidade específica nessa comunicação entre as partes. Alguns dos mais utilizados em aplicações web são:

- GET: requisita dados de um determinado recurso;
- POST: envia informações para o servidor;
- DELETE: envia um comando para remover um recurso específico;
- UPDATE: envia um comando para atualizar um recurso específico.

BROWSER ENGINE

Aqui temos o motor responsável por interpretar todo o código entregue diretamente ao browser, como o HTML, CSS e o Javascript. Cada browser possui a sua própria engine de interpretação, e por isso cada um deles pode ter um comportamento diferente para um mesmo código escrito. Isso acontece muito em códigos javascript modernos, onde uma determinada engine do browser implementa o recurso para suportar uma determinada sintaxe (como arrow functions, spread operator e etc) e outro não.

TRÍADE HTML, CSS & JAVASCRIPT

HTML – HYPERTEXT MARKUPT LANGUAGE

Linguagem de marcação (e não de programação) utilizada para delimitar espaço em documentos de extensão .html. É a linguagem interpretada pelos navegadores e delimita as áreas de cada parte em um documento.

CSS – CASCADING STYLE SHEET

Linguagem declarativa para definição de regras visuais de um documento HTML. Um arquivo HTML pode ter múltiplos arquivos CSS associados a ele. Temos três opções para aplicar uma regra de estilo de CSS a um HTML: arquivo externo, bloco de estilos e inline.

JAVASCRIPT

Linguagem de programação interpretada (pelas browser engines) e fracamente tipada, o que significa que os tipos dos valores atribuídos são inferidos pela variável atribuída. Com o javascript é possível adicionar interatividade às aplicações web, bem como a comunicação com servidores e APIs externas.

FRAMEWORKS E ECOSSISTEMA

No ecossistema de front-end temos vários frameworks que resolvem problemas de formas diferentes. Cada um deles foi criado com um propósito diferente, sendo o React, Angular e Vue os mais conhecidos pela comunidade open-source e mais utilizados pelas empresas aqui no Brasil. Cada um deles possuem características únicas, pois foram criados com propósitos diferentes.

ANGULAR

Pensado para o desenvolvimento de aplicações de ponta-a-ponta, sua instalação inicial já vem com tudo necessário para que quem for desenvolver, foque apenas nas regras de negócio. Isso significa que, para construir um sistema de rotas ou fazer requisições em APIs de forma fácil e consistente, não é necessário a instalação de nenhuma biblioteca de terceiros. O framework ainda possui um sistema de modularização próprio na qual é possível definir um conjunto de *Services/Components/Directives* que fazem parte de um escopo através dos módulos, possibilitando o compartilhamento restrito de funcionalidades dentro de uma mesma aplicação. Ele também utiliza de forma nativa o Typescript, o que, quando bem escrito, garante uma boa escala e organização para a aplicação a ser desenvolvida.

VUE

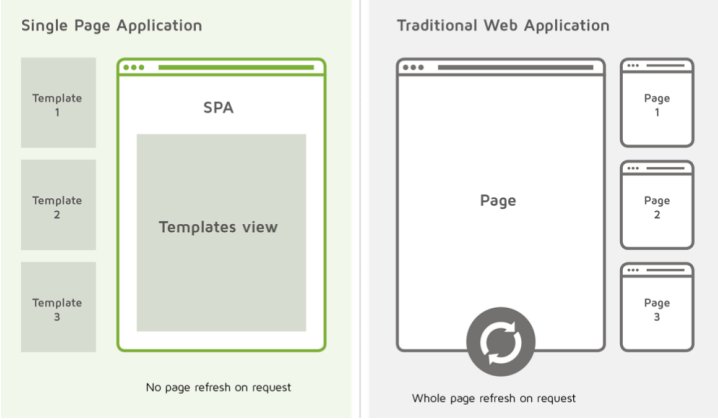
O Vue é um framework totalmente focado em componentes. Isso significa que, sua instalação inicial possui como único recurso a criação de componentes. A proposta do framework é desenvolver interfaces web, através dos componentes, e também possibilitar a convivência com outros frameworks a fim de facilitar a adoção incremental em projetos já existentes. O tamanho em KB (kbytes) da instalação é relativamente pequeno a sua utilização é bem simplificada. Utiliza um conceito chamado de "*single-file componentes*" (componentes em único arquivo), onde tudo o que é necessário para a criação de um componente pode ser escrita no mesmo arquivo (css, view e controller). Muito utilizado em projetos de pequeno porte (o que não inibe sua utilização em projetos complexos) e ideal para quem planeja uma migração gradual de uma base de código legada.

REACT

Talvez o mais popular entre todos, o React foi desenvolvido pelo Facebook com o intuito de também ser um framework orientado a componentes. Nele, não temos nenhuma organização pré-estabelecida e nem um sistema de módulos nativo, ficando a cargo do desenvolvedor definir como será a estrutura da aplicação. Foi um dos primeiros com a abordagem orientada a componentes e utiliza como linguagem o JSX, o que torna possível a escrita de HTML e JavaScript no mesmo arquivo. A principal sacada aqui é que quando vamos escrever algum componente, a lógica quase sempre está atrelada ao que o usuário final percebe, ou seja, a view. Por isso a ideia de escrever ambas as linguagens em um mesmo arquivo possibilita um ganho de produtividade interessante, uma vez que no React, não precisamos aprender nenhuma diretiva customizada para a implementação de loops e blocos condicionais. Podemos utilizar o próprio "if", "while" ou "map" do javascript para implementar estruturas condicionais e de repetição.

ARQUITETURA SINGLE PAGE APPLICATION (SPA)

Em tradução livre, as Aplicações de única página dão uma sensação de maior fluidez para quem navega. Nessa abordagem, ao clicar nos links internos da aplicação, apenas um bloco de conteúdo é recarregado, diferentemente no que ocorre em aplicações comuns (ou Multi Page Application), onde ao clicar em um link interno, toda a página é recarregada para só depois aparecer o conteúdo da página nova. Aqui, nos SPAs, definimos um conjuto de elementos/componentes comuns a todas as páginas e definimos também uma área onde o conteúdo será atualizado. Um exemplo prático é a aplicação do gmail, onde ao clicar em um email, apenas a área central é atualizada e os demais elementos permanecem sempre visíveis ao usuário.



UNIDADE II – INTRODUÇÃO À ARQUITETURA DE APLICAÇÕES

ARQUITETURA DE CSS

Ao desenvolver aplicações web, um ponto importante e que muitas das vezes acaba passando desapercebido é a organização e estruturação do CSS. Diferentemente do Typescript e Javascript, no CSS não temos nenhuma ferramenta que auxilie no rastreamento e refatoração de estilos com tanta eficiência. Isso nos obriga a estruturar muito bem os nossos estilos para que no futuro não tenhamos nenhum problema em adicionar novas funcionalidades e, mais importante, manter os estilos já existentes. Para auxiliar nessa estruturação, temos algumas style-guides e ferramentas que nos direcionam na criação de nomes de classes, definição de propriedades e no ganho de produtividade.

Nome	Style-guide	Ferramenta/lib externa	Compatibilidade com frameworks	Principal característica
BEM	✅	❌	Total	Style-guide para definição de nomes de classes
OOCSS	✅	❌	Total	Style-guide para definição de propriedades de classes
CSS Funcional	❌	✅	Total	Ganho de produtividade e leitura do estado do HTML
CSS-in-JS	❌	✅	React e Vue	Ganho de produtividade e adição de funcionalidades do JS no CSS
Pré-processadores	❌	✅	Total	Ganho de escala e adição de novas funcionalidades ao CSS (variáveis, mixins e etc).

FERRAMENTAS DE BUNDLE E OTIMIZAÇÃO DE CÓDIGO

O bundles (ou empacotadores de código) foram desenvolvidos para automatizar tarefas no desenvolvimento e no build. Coisas como concatenação e minificação de arquivos podem ser feitos de forma automatizada, bem como a execução de testes e adição de referência de arquivos no HTML. Todos os frameworks modernos de desenvolvimento web utilizam algum bundle por trás. Dentre as várias opções que temos, o **webpack** é o mais utilizado. Com ele, conseguimos processar arquivos SASS e definir regras para qualquer tipo de extensão de arquivo, como o typescript. Entender o seu funcionamento nos dá a possibilidade de customizar tarefas de desenvolvimento de acordo com a necessidade e especificidade da nossa arquitetura.

ARQUITETURA MODULAR PARA APLICAÇÕES ESCALÁVEIS

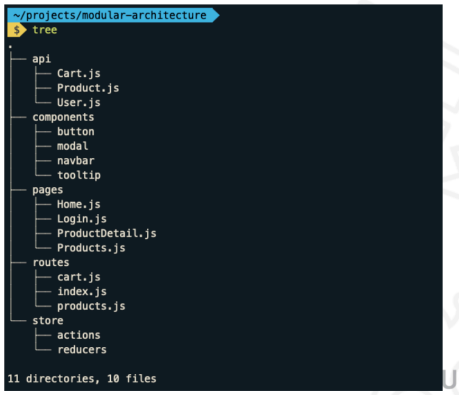
O objetivo dessa estruturação é garantir que nossa aplicação ganhe escala independente do rumo que ela tome. Possibilitar a aplicação de um dos conceitos da arquitetura evolutiva chamado de "*last responsible moment*" é também um dos focos, uma vez que desenvolver aplicações que possibilitam postergar de forma inteligente decisões complexas é de suma importância. A arquitetura modular no front-end deve ser independente de frameworks e possibilite um ganho de escala sem prejudicar a organização dos arquivos e suas responsabilidades. Nessa arquitetura, devemos nos ater a definição das responsabilidades de cada módulo bem como os arquivos que os compõem:

1. **UI Components (ou smart componentss)**: componentes que concentram a lógica visual e de interação com o usuário (botões, menus, listas, datepicker, modal...);

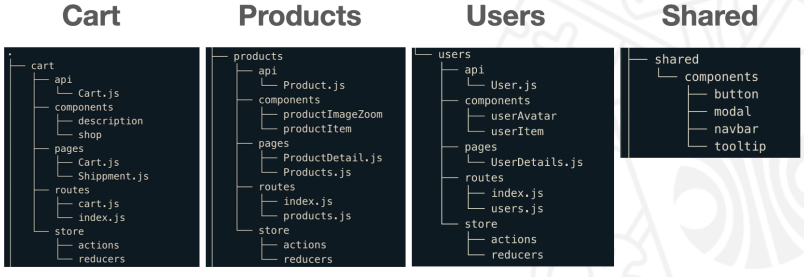
2. **Page components (ou view/container components):** componentes que representam uma tela e estão associados a uma rota. Concentram também as requisições às APIs e funcionam como um "agregador" de vários UI Components;
3. **Application Routes:** camada de mapeamento de rotas e componentes. Cada rota deve nos levar a um componente e a definição dessas regras não deve estar misturada a outras regras de negócio da aplicação, uma vez que elas podem crescer exponencialmente junto com a aplicação como um todo;
4. **API Client/Services:** camada que concentra a lógica necessária para requisição à APIs como definição de cabeçalhos, URLs base e verbos das requisições (POST, GET, PUT...). Ponto único de contato entre qualquer componente e a API externa.
5. **State management:** parte do projeto que concentra as regras de gerenciamento de estado da aplicação. Aqui temos a possibilidade de compartilhar estados com todos os componentes e utilizar esse benefício a nosso favor. Utilizaremos a Arquitetura Flux para a estruturação do nosso módulo de gerenciamento de estados.

Um ponto importante é que cada módulo pode conter sua própria estrutura com base na descrição acima. Caso o projeto seja de pequeno porte, a divisão exemplificada pode abranger todo o escopo da aplicação. Ou seja, em aplicações de médio/grande porte, podemos ter um módulo "Produtos" que possuam os módulos UI Components, Page Components e etc. Em aplicações menores, para não adicionarmos uma complexidade desnecessária, podemos considerar os módulos como sendo os UI Components, Page Components e todos os demais e, dentro de cada um deles, colocamos todos os arquivos da aplicação que se encaixarem nesse contexto.

Arquitetura modular para aplicações escaláveis - Opção 1

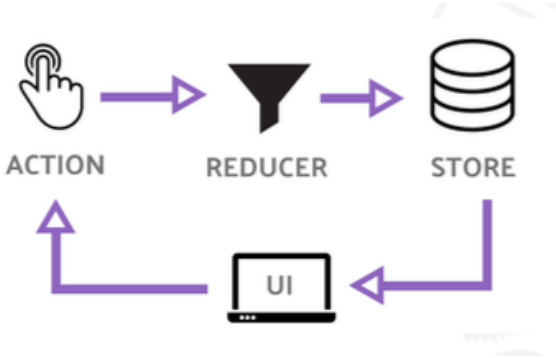


Arquitetura modular para aplicações escaláveis - Opção 2



ARQUITETURA FLUX

A ideia dessa implementação é direcionar o projeto ao criar interfaces de usuário que dependam de gerenciamento de estados a nível de aplicação. Aqui, trabalhamos com basicamente três conceitos, representados pela imagem abaixo:



Fonte: https://medium.com/reactbrasil/iniciando-com-redux-c14ca7b7dcf

STORE

Armazena o estado da aplicação e é atualizado por um dispatcher (que emite uma action). Toda aplicação deve possuir uma única store.

ACTIONS

Conteúdos enviados da camada "UI" (ou qualquer componente) que descrevem **o que** acontece. Aqui, escrevemos uma função javascript que retorna um objeto. Este objeto contém o payload (caso necessário) e um type para identificar o que a ação pretende fazer. Não é comum adicionarmos nenhuma lógica aqui, apenas a estruturação da ação por meio de funções/objetos.

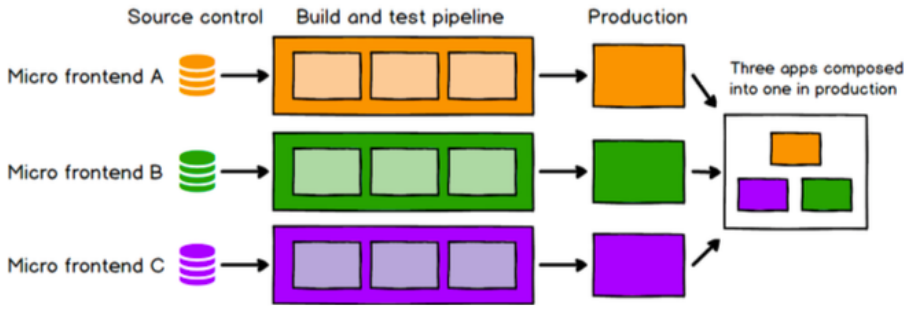
REDUCERS

Descreve **como** as alterações acontecem. Aqui, um reducer recebe uma action como argumento em uma função e sempre concentra a lógica necessária par atualizar a store. Exemplo: se uma action descreve um novo objeto em um array de produtos, o reducer deve implementar essa action escrevendo uma função para adicionar um objeto em um array (utilizando um *map*, *filter*, *for* ou qualquer estrutura do javascript).

UNIDADE III – ABORDAGENS ARQUITETURAIS

ARQUITETURA DE MICRO FRONT-ENDS

Na medida em que as aplicações front-end têm asumido cada vez mais responsabilidades dentro do contexto de uma organização e ficado cada vez mais complexas, surgem abordagens que tentam solucionar problemas de escala e garantir ainda mais independência entre as várias partes que compõem um projeto. A principal ideia aqui é conseguir ter uma maior independência entre módulos com uma proposta independente de frameworks. A lógica pode ser pulverizada em vários projetos diferentes e a pipeline de build, test e deploy também deve ser independente. É sempre importante ponderar a complexidade que estamos dispostos a manter em uma aplicação. Utilizar a abordagem de micro front-ends sem a necessidade pode trazer uma alta complexidade desnecessariamente nas nossas aplicações.



Fonte: https://martinfowler.com/articles/micro-frontends.html

ARQUITETURA DE APLICAÇÕES EM MONOREPOS

Abordagem arquitetural que propõe um repositório para múltiplos pacotes. Cada pacote representa um projeto (podendo ter o seu próprio controle de versão) e a possibilidade de isolamento de código. Aqui, também conseguimos aplicar a estratégia de micro front-ends e além disso, facilitar o gerenciamento de configurações globais (o que é extremamente desafiador quando se separa um projeto em vários repositórios). Essa abordagem já foi implementada em projetos relevantes como o próprio React, onde cada uma de suas dependências é implementada no mesmo projeto e entregue em pacotes diferentes no NPM.

Fonte: https://www.toptal.com/front-end/guide-to-monorepos

Algumas ferramentas como o Lerna e o Yarn Workspaces auxiliam na implementação dessa abordagem.

APLICAÇÕES SERVER-SIDE RENDERING (SSR)

A arquitetura de single-page application trouxe inúmeros benefícios, mas também trouxe alguns desafios. Dentre eles temos o SEO (*Search Engine Optimization*). Nos SPAs, as aplicações são renderizadas e interpretadas no próprio browser através dos scripts, o que dificulta as ferramentas de busca a saberem do que se trata aquela página. Nesse caso, quando queremos utilizar uma abordagem moderna mas que também seja eficaz nos resultados de busca, podemos recorrer à abordagem Server-side rendering, onde o conteúdo é renderizado no servidor e a página já possui o HTML estruturado na requisição GET no recurso. Além disso, eliminamos uma boa parte do TTI (*time to Interact*) pois nos SPAs, boa parte do tempo de carregamento da página deve-se ao fato de que o browser precisa renderizar todos os elementos da página após carregar os scripts, o que não acontece no SSR pelo fato de que a página já vem prontamente renderizada.

UNIDADE IV – FUNDAMENTOS DE SEGURANÇA E UM OLHAR FORA DA CAIXA

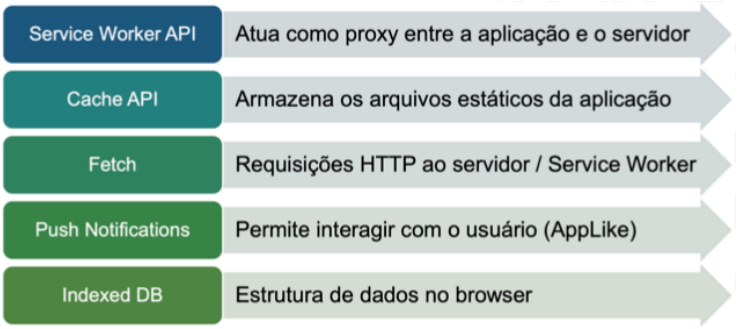
SERVERLESS COMPUTING

Com a computação serverless (sem servidor) temos a possibilidade de execução de funções sem a necessidade de subir e configurar todo um servidor ou mesmo um projeto por inteiro para tal. Nesse modelo também temos suporte a inúmeras linguagens (não somente ao Javascript) e uma escala bem interessante para resolver alguns problemas específicos. A cobrança é baseada em sua execução e performance. Sendo assim, se temos alguma função implementada que não estamos utilizando, não seremos cobrados pelo código implementado e toda a infra necessária para rodá-la. Pense no exemplo de emissão de relatórios mensais em uma dashboard. Essa função provavelmente concentra uma certa complexidade que pode ficar em "stand-by" durante todo o mês e ser "ativada" automaticamente quando precisarmos de gerar algum relatório mensal. Um outro caso de uso comum é o seu uso em IoT (internet of things). Algumas das plataformas que suportam tal aplicação são:

- AWS Lambda;
- Azure Functions;
- Google Cloud Functions;
- Netlify Functions.

APLICAÇÕES PWA (PROGRESSIVE WEB APPLICATIONS)

"Os Progressive Web Apps fornecem uma experiência instalável, semelhante a um aplicativo, em computadores e dispositivos móveis que são criados e entregues diretamente pela Web. Eles são aplicativos da web que são rápidos e confiáveis. E o mais importante, são aplicativos da web que funcionam em qualquer navegador. Se você está construindo um aplicativo da Web hoje, já está no caminho da criação de um aplicativo da Web progressivo." (Progressive Web Apps – Web.dev). Nessa abordagem, o código javascript pode ser executado em background (mesmo com a aba do navegador fechada) afim de possibilitar o envio de notificações push e várias outras features off-line. Com os PWAs conseguimos ter uma experiência muito próxima do nativo, pois muitas das funcionalidades podem ser executadas mesmo sem acesso a internet. Nos PWAs podemos utilizar o conjunto de APIs abaixo para manipular eventos e interceptar requisições do browser:



WEB ASSEMBLY

Um tipo de código binário que consegue ser executado em browsers modernos consegue extrair em muito o potencial de algumas aplicações. Aqui, temos a possibilidade de escrever código em C#, por exemplo, e compila-lo para um binário Wasm (Web Assembly). Alguns dos casos de uso mais vistos no mercado são:

- Jogos 3D;
- Realidade aumentada e virtual;
- Edição de imagem/vídeo (ver caso de uso do Figma).

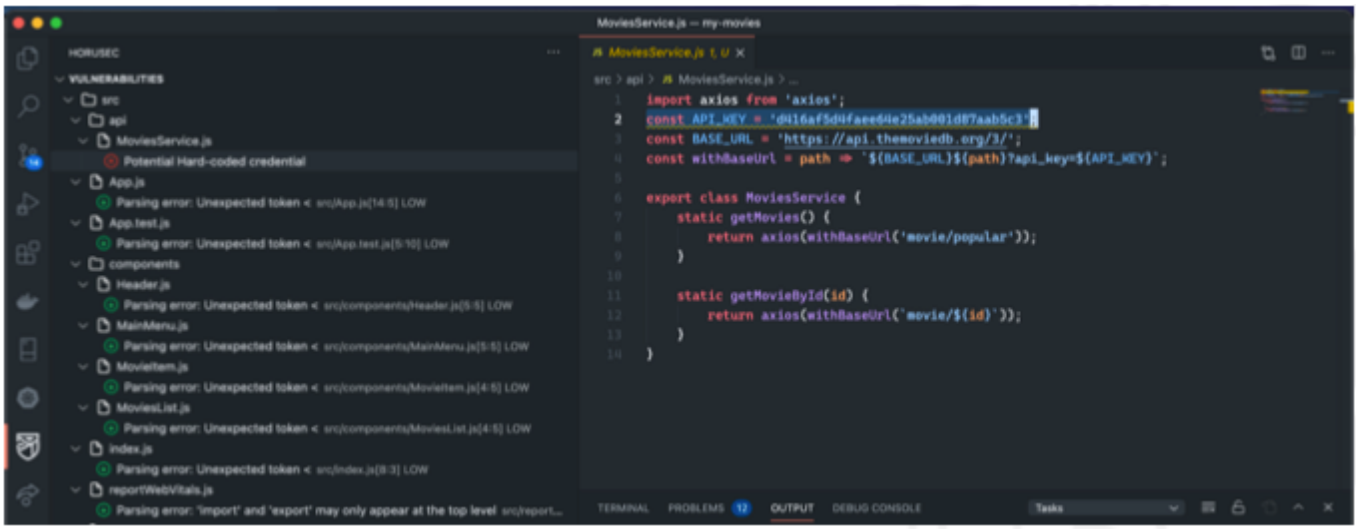
A Microsoft têm investido em frameworks para possibilitar a construção de aplicações Web Assembly através do Blazor, um projeto já em evolução que permite utilizar vários recursos da web através do C#.

FUNDAMENTOS DE SEGURANÇA EM FRONT-END

Para trabalharmos com os gaps de segurança em aplicações front-end podemos utilizar basicamente duas esferas de conhecimento: boas práticas e ferramentas. Nas boas práticas, seguimos as orientações da OWASP Top Ten para implementar códigos defensivos e que previnem uma série de ataques por meio de validações relativamente simples. As principais brechas exploradas com relação à aplicações front-end são:

- Injection;
- Cross-Site Scripting;
- Using Components with Known Vulnerabilities;
- Insufficient Logging & Monitoring.

Dentre as inúmeras **ferramentas** disponíveis, uma alternativa interessante e open-source é a Horusec. Com ela conseguimos identificar vulnerabilidades no código em tempo de desenvolvimento



Outras duas ferramentas utilizadas com essa finalidade são:

- NPM audit: comando do npm para identificar dependências com vulnerabilidades conhecidas;
- Verdaccio: servidor open-source para armazenar módulos do javascript sem a dependência do repositório público do npm. Com ele, conseguimos hospedar nossos códigos em um servidor próprio e aplicar práticas de segurança ainda mais efetivas e controladas.