Universidade de Brasília Faculdade UnB Gama

Disciplina: Desenvolvimento Avançado de Software

Prof.: André Lanna

Framework: Quasar + Vue.js

Cecilia Dib - 14/0134425

Danilo Barros Mendes - 12/0114780

Naiara Andrade - 15/0043449

Thiago Canabrava Moreira - 14/0163859

Vitor Bertulucci Borges - 14/0171126

Brasília, 27 de Junho de 2018

O que é framework?

Framework, em software, é um sistema ou aplicação baseado em extensibilidade, composto por um conjunto de classes que suportam certas funcionalidades mas permitem pontos de adaptação/extensão. Frameworks também se baseiam em reutilização, onde os desenvolvedores desses softwares devem estar concentrados em criar códigos simples e genéricos para possibilitar que a ideia de reutilização seja seguida, contemplando funcionalidades que podem ser reutilizadas em diversas outras aplicações.

Em outras palavras, Framework possibilita a solução para um ou mais problemas semelhantes usando um conjunto de classes e/ou interfaces que possibilitam a decomposição desses problemas por meio de sua flexibilidade e extensibilidade, diminuindo, assim, o esforço na criação das aplicações as quais possuem as funcionalidades implementadas pelo framework. Dessa, forma pode-se ver um framework como uma aplicação “semi-completa”, onde é possível “completá-la”, ou à especializar com trechos de código externos (FAYAD et. al., 1997).

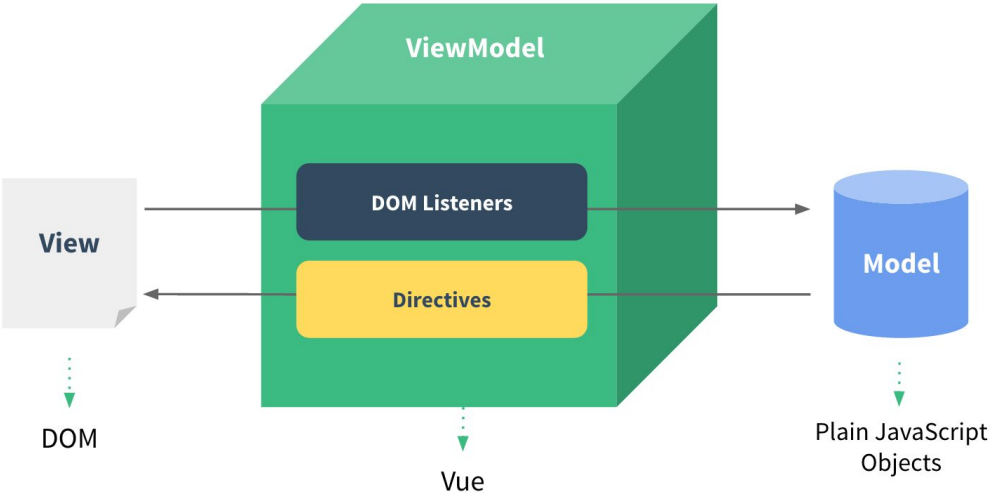
Qual o framework escolhido e qual o seu propósito (desenvolvimento web, mobile, front-end)?

O Quasar é um framework proposto para auxiliar o desenvolvimento de aplicações, sejam elas *websites, Progressive Web Apps* (PWA), aplicações móveis (com auxílio do Apache Cordova) e aplicativos desktop (utilizando o Electron), todas para o desenvolvimento *front-end*. Ele utiliza em sua estrutura o Vue.js, que é um framework progressivo para a construção de interfaces de usuário, baseado no webpack, que é um empacotador javascript.

Como ele é estruturado (há diagrama de classes, arquitetural, dentre outros)?

Pelo Quasar ser um framework relativamente novo (lançado em Agosto de 2016 e atualmente na versão 0.16) e com uma comunidade pequena, porém crescente, não existe um diagrama de classes ou arquitetural oficial do projeto. Apesar disso, o Quasar é baseado no Vue.js, que tem sua arquitetura bem definida em sua documentação.

O Vue.js utiliza em sua arquitetura o padrão MVVM que é uma especialização da arquitetura MVP (comunicação bidirecional com as outras camadas). No MVVM, ocorre comunicação entre camadas através de data binding e há restrições de acesso entre as camadas. Mais especificamente, o Vue é focado na camada ViewModel deste padrão arquitetural, onde o mesmo conecta a camada de Visualização e a camada de Modelo por meio de ligações de dados bidirecionais. Na camada Model, utilizando javascript, são gerados objetos que resultam em manipulações em tempo real no DOM. Toda parte de UI compilada é mostrada na camada visual.



Arquitetura MVVM do Vue.js.

Brasília, 27 de Junho de 2018

Como citado anteriormente, o Quasar segue uma arquitetura similar à do Vue.js, então uma pode ser explicada através da outra. A única diferença a se citar, é que, no Quasar, toda a configuração da sua aplicação é feita através do quasar.conf.js, seja ela relacionada à CSS, build, host, webpack, rotas, deploy, dentre outros. No Vue.js estas configurações são feita em vários diretórios e arquivos diferentes, mas todas elas têm o mesmo objetivo das configurações unificadas no Quasar.

Quais são os pontos de extensão de funcionalidades desse framework (hot-spots)?

Os pontos de extensão encontrados no Quasar são diversos, visto que o mesmo é construído baseado em um framework progressivo, o qual é desenhado desde o início para ser incrementalmente adaptado ao contexto o qual está sendo utilizado.

Dessa forma podemos listar o próprio arquivo *.vue* como um ponto de extensão do framework, onde a partir de regras bem estabelecidas pode-se construir um código externo que se execute e se comporte como parte do framework. Um exemplo de um arquivo *.vue* pode ser visto abaixo, onde logo ao início, a *tag* <*template*>, do arquivo é reservado para comportar códigos HTML, seguidos de códigos Javascript (*tags* <*script*>) e CSS (*tags* <*style*>). Pode ser visto também, que dentro da seção designada para códigos Javascript uma estrutura de exportação das funcionalidades do código devem ser exportadas.

```
<template>
 <!-- you define your Vue template here -->
</template>

<script>
// This is where your Javascript goes
// to define your Vue component, which
// can be a Layout, a Page or your own
// component used throughout the app.

export default {
 //
}
</script>

<style>
/* This is where your CSS goes */
</style>
```

Esqueleto de arquivo *.vue*.

Já para o Quasar pode-se citar o uso dos componentes como um ponto de *hot-spot*, onde o usuário invoca o componente de sua escolha e a partir das diretrizes definidas previamente por tal componente, ele é capaz de inserir um código customizado para a construção da sua aplicação. Um exemplo de tal uso pode ser visto abaixo, onde o usuário invoca o componente "*q-btn*" e passa como parâmetro uma função ("*doSomething*") customizada para que seja executada uma vez que certo comportamento for invocado pelo próprio framework.

```
<div>
 <q-btn @click="doSomething">Do something</q-btn>
 <q-icon name="alarm" />
</div>
```

Componente do Quasar *q-bnt* sendo utilizado.

Devido os frameworks citados serem baseados em Node.js, toda a biblioteca NPM (biblioteca de pacotes reutilizados feitos em javascript) é compatível. Quem os

utiliza para desenvolvimento pode facilmente acoplar um pacote externo à sua aplicação. Um exemplo é a capacidade de utilizar template engines para compilar html. Apenas utilizando o comando yarn add pug ou npm install --save pug, o desenvolvedor é capaz gerar html utilizando o pug. Após instalado, basta utilizar <style lang="pug">, onde a flag *lang* dá suporte à utilizar qualquer tipo de template engine criada por terceiros ou até mesmo por você.

Qual o modelo de componente definido para o framework? Ele está publicamente definido e acessível?

Quasar é uma estrutura de código aberto licenciada pelo MIT (desenvolvida com o Vue.js) que ajuda os desenvolvedores da *web* a criar:

- *Websites* responsivos;
- PWAs (*Progressive Web App*);
- Aplicativos móveis (Android, iOS) por meio do Apache Cordova;
- Aplicativos de *Desktop* multiplataforma (usando o Electron).

O Quasar permite que os desenvolvedores escrevam códigos uma vez e simultaneamente os implantem em formato *website*, PWA, *Mobile App* e/ou Electron App usando a mesma base de código. Assim como, permite a possibilidade de projetar um aplicativo em tempo recorde, usando um CLI de última geração e apoiado por componentes da *Web* Quasar muito rápidos e bem escritos.

Com o Quasar, não é necessário utilizar bibliotecas adicionais como Hammer.js, Moment.js ou Bootstrap.

Uma fonte autorizada de código para todas as plataformas, simultaneamente: site *desktop*/móvel responsivo, PWAs (Aplicativos *Web* Progressivos), aplicativos móveis (que parecem nativos) e aplicativos de *desktop* multiplataforma (por meio do Electron).

Há um componente para quase todas as necessidades de desenvolvimento da *web*. Cada um desses componentes é cuidadosamente criado para oferecer a melhor experiência possível aos usuários. O Quasar é projetado com desempenho e

capacidade de resposta em mente - então a sobrecarga de usar o Quasar é quase imperceptível.

Os desenvolvedores que usam o Quasar são encorajados a seguir as práticas recomendadas de desenvolvimento da *Web* e vêm incorporados com muitos desses recursos prontos para uso. Pode-se citar alguns como, Minificação HTML / CSS / JS, *cache busting*, *tree shaking*, *sourcemapping*, *code-splitting* e carregamento lento, *transpiling* ES6, *linting code*, recursos de acessibilidade, suporte RTL (da direita para a esquerda) para os componentes Quasar e para o próprio código do desenvolvedor (caso o site/aplicativo escrito pelo desenvolvedor use um pacote de idiomas RTL, o código CSS é convertido automaticamente para tal).

O Quasar oferece uma versão UMD (*Unified Module Definition*), o que significa que os desenvolvedores podem adicionar uma *tag* CSS, JS e HTML em seu projeto existente e eles estão prontos para usá-la. Nenhuma etapa de compilação é necessária.

Ao usar o CLI do Quasar, os desenvolvedores se beneficiam de:

- Estado preservação *hot-reload* ao fazer alterações no código-fonte do aplicativo, não importa se é um site, PWA, um aplicativo móvel (diretamente em um telefone ou em um emulador) ou um aplicativo eletrônico. Os desenvolvedores simplesmente alteram seu código e podem assisti-lo na atualização, sem a necessidade de qualquer atualização de página;
- Estado que preserva a sobreposição de erro de compilação;
- *Lint-on-save* com ESLint - se os desenvolvedores gostam de aplicar *linting* ao seu código apenas;
- *Transpiling* código ES6;
- Mapas de origem;
- Alterar as opções de construção não requer uma recarga manual do servidor de desenvolvimento;
- Muitas outras ferramentas e técnicas de desenvolvedor de última geração.

Portanto, para que tudo isso seja possível uma boa organização interna é necessária. Dessa forma, o framework Quasar utiliza de boas políticas de versionamento para armazenar seus componentes, os quais, juntamente com seus respectivos comportamentos e modelos estão definidos em um único diretório em seu repositório (*link* para o diretório componentes).

Dessa forma, é possível notar que o padrão de definição dos componentes se baseia na definição de comportamento utilizada pelo framework Vue.js, no qual define campos para a seção de exportação do código Javascript de um componente. Além dos comportamentos únicos de cada componente o Quasar adiciona dois arquivos de estilo, no qual definem o estilo visual dos componentes sobre situações diferentes, que podem ser controladas e configuradas pelo desenvolvedor.

Além disso, um arquivo de configuração (components.js) é utilizado para expor todos os componentes necessários, para que então estes possam ser utilizados pelo desenvolvedor.

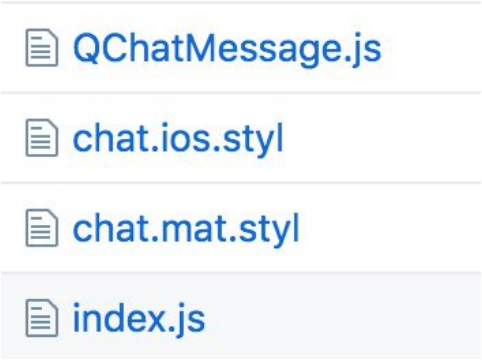
O que é um componente de software para o framework adotado e como ele é estruturado?

Para o Quasar, um componente de software é definido utilizando as diretrizes do arquivo *.vue* (apesar de definir seus componentes em arquivos *.js*), onde parâmetros como métodos definidos, variáveis fixas e mutáveis, assim como o próprio nome do componente a ser criado.

```
import ...
export default {
 name: 'ComponentName',
 props: {
 },
 computed: {
 },
 render (h) {
 },
 methods: {
 }
}
```

Padrão de componente do Quasar.

Além disso, o framework define arquivos de configuração dos estilos (*.styl*) dos seus componentes para que possam ser utilizados em uma compilação para dispositivos móveis. Sendo assim, a arquitetura final de um diretório comum para um componente pode ser vista na imagem abaixo.



Exemplo de um diretório de um componente.

Como os detalhes de sua implementação são ocultados dos framework?

Para o ocultamento da implementação dos componentes originários do framework Quasar, o próprio Vue.js toma conta da possível indireção necessária à serem feita. Dessa forma, métodos que necessitam de privacidade são declarados com um "_" (*underline*) precedendo a sua nomeação. Com isso a implementação de tal método se torna oculta dos demais que o forem invocar. Além disso, campos

como “`render (h)`” são inerentemente privados ao componente, já que definem comportamentos de inicialização gráfica de um componente e seus dependentes, antes dele ser mostrado na tela da aplicação.

Como suas interfaces de comunicação / composição devem ser definidas?

Para as interfaces de comunicação o Quasar se apoia fortemente nas definições do Vue.js, que já tem por definição propriedades públicas (*props*), eventos definidos de componentes (Ex: @*click*), troca de mensagem entre componentes (*emit*) e funcionalidades reutilizáveis (*mixin*). Todas essas citadas são utilizadas para comunicação entre componentes, porém ainda existem as interfaces de comunicação dos componentes com o framework, onde existe um gerenciamento de eventos para seu ciclo de vida.

Quais são os modos geralmente utilizados para composição componentes / frameworks? Como componentes de software são adaptados para o framework sob investigação? Glue-code? Herança? Adapter? etc…

Quasar Apps são construídos com blocos de alto nível que por convenção são chamados componentes. Os componentes permitem criar rapidamente uma interface para o aplicativo, que pode ser utilizado por toda a aplicação, utilizando apenas um simples *import* para outros componentes criados. O Quasar vem com vários componentes defaults, incluindo modais, folhas de ação, colapsíveis, cartões, diálogos, FAB, listas, entre outras.

Os componentes do Quasar são gravados como *Web Components*, portanto, incorporam códigos HTML, CSS e Javascript que você pode usar incluindo apenas uma tag HTML nos modelos de página e layout.

O Vue.js incentiva o uso de componentes para encapsular códigos reutilizáveis e, portanto, implementar o conceito DRY (*don’t repeat yourself*). O Quasar também segue essa prática e distribui todos os seus componentes

encapsulados.

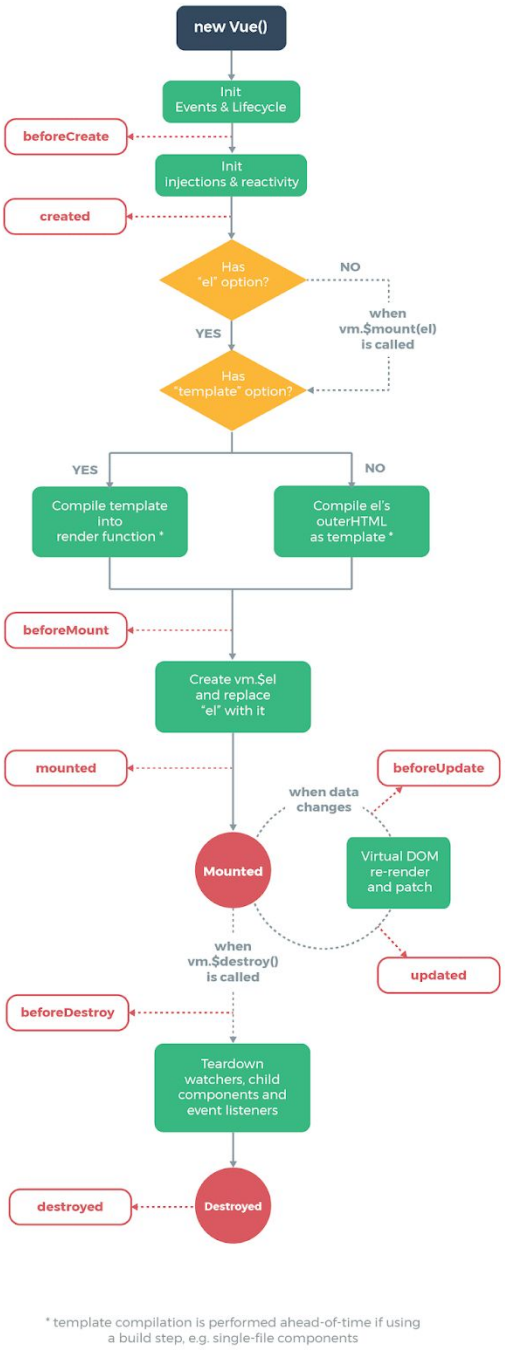
O Quasar é uma estrutura e, portanto, fornece blocos de construção para criar seus próprios aplicativos. O Quasar e o Vue.js utilizam **Herança**, que é um conceito conhecido de programação orientada a objetos, onde classes são capazes de estender outra classe para reutilizar seus métodos e atributos para construir uma classe nova, mas similar. A composição, por outro lado, também é conhecida da programação orientada a objetos, mas em vez de estender ou sobrescrever uma classe existente, a classe usa outras classes para fornecer alguns serviços comuns. Ao instanciar um componente que foi criado, é feita uma relação de extensão do componente raiz, sendo sua derivação

O padrão ***Decorator/Wrapper*** é conhecido pela programação orientada a objetos e permite criar novas versões de um objeto e alterar ou estender seu comportamento envolvendo o objeto existente e dando-lhe um novo nome, não afetando, portanto, o objeto base. Também é utilizado as diretivas como forma de reusar código.

Como se dá a comunicação entre componentes / framework? Há um diagrama comportamental (sequência, colaboração, etc) que demonstre tal comunicação?

Pelo fato do Quasar ser baseado fortemente no Vue.js e em cima das suas decisões de framework, como comunicação interna, a interface componente-framework utilizada é a original do Vue.js. Portanto, é definido pelo Vue.js, que os componentes devem seguir um ciclo de vida fixo, onde em certos pontos eventos específicos são invocados.

Os eventos presentes no ciclo de vida de um componente podem ser vistos na imagem abaixo, onde temos as emissões de mensagens quando ocorrem os retângulos vermelhos (Ex: *beforeCreate*).



Ciclo de vida de um componente e suas mensagens.

Dessa forma, a função de cada um desses eventos pode ser explicada de forma sucinta.

Brasília, 27 de Junho de 2018

- ***beforeCreate()***: Esse *callback* para o evento é onde se começa o fluxo de chamadas do Vue.js. Não acontece muita coisa nessa fase, apenas é considerado como o início da execução do programa;
- ***created()***: Uma vez que a instância do Vue.js está sendo criada, a maioria das tarefas relacionadas a observação do dado já é automaticamente acobertada por esse evento. Essa chamada é feita pelo framework após a observação das propriedades mutáveis (*computed*), métodos (*methods*) e métodos observadores (*watchers*);
- ***beforeMount()***: Essa chamada é feita apenas para a compilação da seção *template*, onde imediatamente após tal tarefa a próxima chamada de evento é feita;
- ***mounted()***: Logo após a chamada de *beforeMount* é completada, o *template* (que retêm toda a definição do modelo de objeto de documentos virtuais, *Virtual DOM*) é montado utilizando a instancialização de processos do framework. Visto isso, essa fase é onde o seletor de elementos de CSS é criado e é onde os dados estão prontos para ser conectados no *DOM*;
- ***beforeUpdate()***: Esse *callback* é chamado junto ao *updated*, quando existem mudanças ocorrendo no *DOM*. Sendo assim, esse evento pode ser chamado também logo após a fase *mounted*;
- ***updated()***: É acionado para renderização de eventos no DOM. Por exemplo, ao clicar em um botão e um valor é atualizado, a mudança no valor precisa ser mostrada no DOM, e para isso este método é chamado;
- ***beforeDestroy()***: Acionado automaticalmente pelo Vue quando o componente está pronto para ser destruído;
- ***destroyed()***: Última parte do ciclo de vida, onde libera todas as diretivas, métodos e observadores. Após isso, todos os elementos citados são encerrados, e sua instância fechada.

Referências

1. Fayad, Mohamed, and Douglas C. Schmidt. "Object-oriented application frameworks." *Communications of the ACM* 40.10 (1997): 32-38.
2. Tutorialsplane. Vue.Js LifeCycle Diagram. (2017) Disponível em: <https://www.tutorialsplane.com/vue-js-lifecycle-diagram/>. Acesso em: 26 de Junho de 2018.
3. Vue. A instância Vue (2017) Disponível em: <https://br.vuejs.org/v2/guide/instance.html>. Acesso em: 26 de Junho de 2018.
4. Quasar Documentation (2017) Disponível em: <https://quasar-framework.org/guide/>. Acesso em: 26 de Junho de 2018.
5. Vue.js, Introduction (2016) Disponível em: <https://012.vuejs.org/guide/#Introduction>. Acesso em 26 de Julho de 2018.