Projet WAVES : Des flux de données brutes et hétérogènes à l'information qualifiée

N° du contrat F1411006 Q Date de début 2 juin 2014 Durée 36 mois

Livrable D5.2
Dataset et Datastream Visualisation : Composants de visualisation

1 Statut

Niveau dissémination	Publique
Date d'échéance	M16
Date de soumission	M21 25/03/2016
Work Package	5
Tâche T	Dataset et Datastream Visualisation
statut d'approbation	Draft
Version	0.1
Nombre de Pages	1
Nom du fichier	D5.2 - Composants

2 Historique

Version	Date	Revu par
0.0	11/03/2016	-
0.1	25/03/2016	

3 Auteurs

Organisation	Nom	Contact
Data-Publica	Loïc Petit	loic.petit@data-publica.com

4 Résumé

Le but de ce livrable est de rendre compte des travaux effectués dans l'étude technologique pour rendre la visualisation des flux de données la plus complète et flexible possible.

Nous présenterons ici les maquettes du tableau de bord que nous cherchons à développer dans cette approche, ensuite nous aborderons les aspects technologiques en détaillant les frameworks utilisés et enfin leur application dans notre contexte.

Contenu

5 Introduction

Notre approche dans ce livrable est légèrement différente du but initial proposé lors de l'annexe technique pour la raison suivante : beaucoup de primitives de visualisation de données existent actuellement et son utilisables telles quelles. Toutefois, de notre point de vue, après étude, nous remarquons que le composant n'est pas le problème en soit, c'est la composition de l'application qui rend difficile la réutilisation des composants.

Nous détaillerons donc les technologies que nous avons mises en œuvre dans jusqu'ici pour composer notre application dans le but de réutiliser au maximum les composants. Cette approche a été utilisée dans la conception du prototype v0.

6 Tableau de bord composable

6.1 Application au cas d'usage

Il nous a été précisé après plusieurs discussions avec Ondéo Systems que le but de Waves ne devait pas être de reproduire AquaAdvanced. Le principe est de chercher de nouvelles données et de les croiser massivement grâce au support de l'ajout de contexte. Des exemples pourraient être les suivants : variation du trafic routier, agenda des fêtes, météo, circulation trafic, réseaux sociaux...

Le constat que cela impose sur le produit final est le suivant : on souhaite explorer les données mais : on ne connaît pas actuellement l'ensemble des possibilités (actuelles ou futures). Le point de vue des concepts que nous avons présenté dans le rapport 5.1, le tableau de bord composable reste valide.

Enfin, le tableau de bord permet, comme expliqué précédemment, d'établir une vue d'ensemble d'un système. Le problème est de supporter le caractère exploratoire qui est normalement laissé aux visualisations dédiées. Nous proposons de résoudre ce problème en permettant une grand configurabilité du tableau de bord.

Le principe étant de placer des widgets sur un page de façon dynamique et chaque emplacement pourra être lui même être configurable.

6.2 Scénario et maquettes

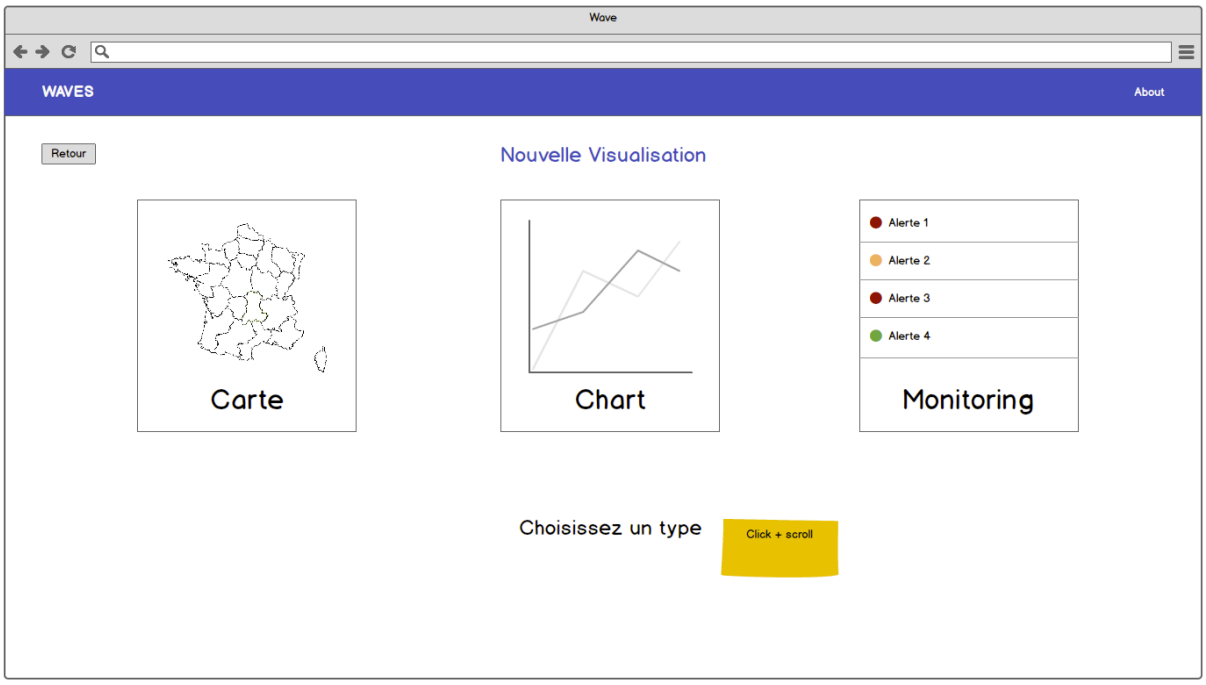
Le scénario principal de l'application se constitue ainsi :

1. La page d'accueil permet d'ajouter un widget grâce au bouton « + » [voir figure 1]
    a. L'utilisateur clique donc sur le bouton

Figure 1 - Page d'accueil

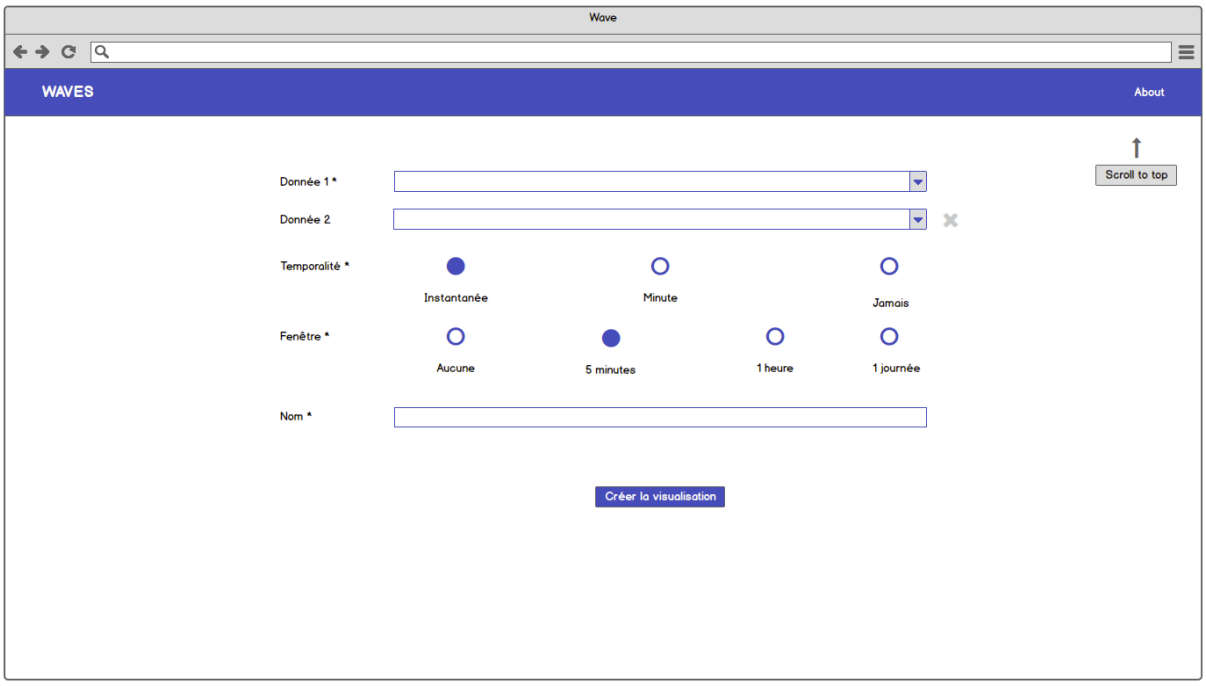
2. Plusieurs choix lui sont présenté [voir figure 2]
    a. Les visualisations principales sont mises en avant (ici nous proposons Carte, Line Chart et Monitoring)
    b. L'utilisateur sélectionne un type de visualisation

Figure 2 - Sélection d'un type de visualisation



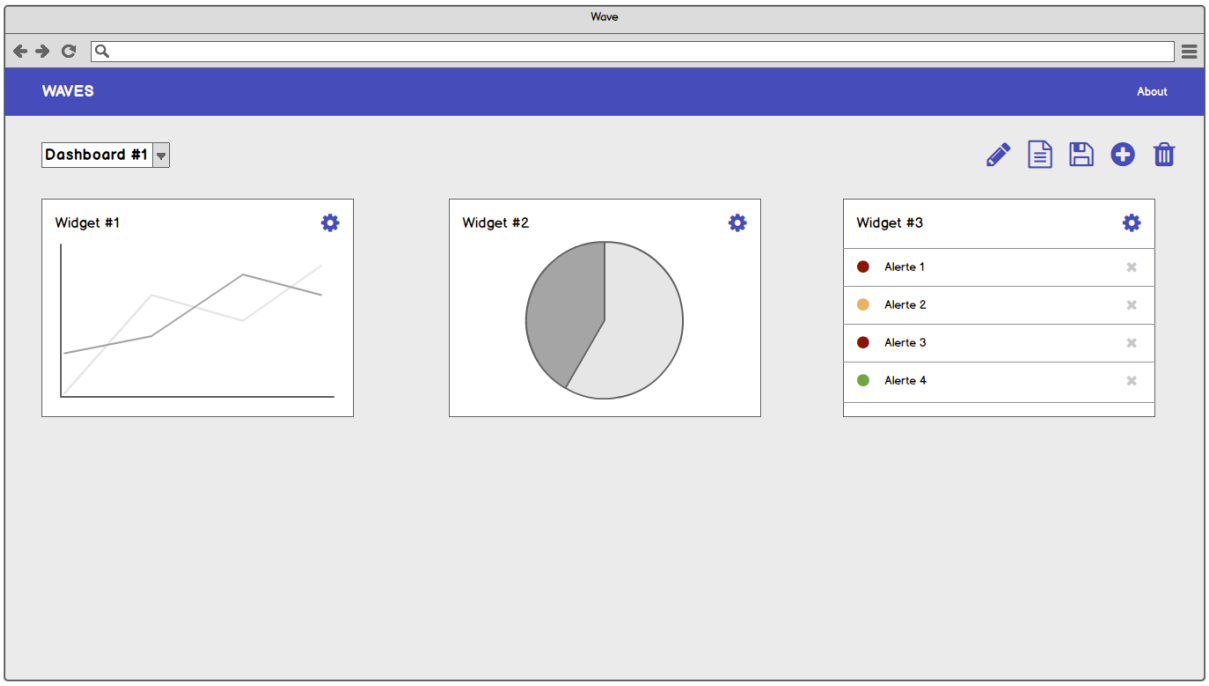
3. Un panel en plein écran permet à l'utilisateur de choisir ses options de visualisation [voir figure 3]. Les options sont les suivantes :
    a. Donnée 1 : Sélectionner parmi les propositions la donnée à afficher (par exemple waves:numericValue)
    b. Donnée 2 : Sélectionner la deuxième donnée à comparer (le timestamp par exemple)
    c. Temporalité : permet de sélectionner la dynamique voulue sur la donnée
    d. Fenêtre : permet de sélectionner la fenêtre de données voulue, ceci pourrait être changeable aisément
    e. Nom : Permet d'associer un label à la visualisation pour la retrouver.

Figure 3 - Configuration



4. Répéter l'étape d'avant pour concevoir l'ensemble des visualisations voulues
5. Consulter le tableau bord qui est mis à jour en fonction de la périodicité voulue avec les fenêtres d'exécutions demandées [voir figure 4]

Figure 4 - Tableau de bord une fois configuré



Comme présenté dans la figure 4, l'utilisateur pourra constituer un ensemble de tableau de bord qu'il pourra consulter en fonction de ses besoins. Il pourra aussi éditer chaque tableau de bord (ajouter, renommer, bouger ou supprimer une cellule par exemple).

Pour mettre en œuvre ce tableau de bord, nous utilisons un ensemble de technologies *frontend* que nous allons désormais présenter.

7 Détail des technologies

7.1 TypeScript

TypeScript est un langage de programmation libre et open-source développé par Microsoft couvrant JavaScript. Ce langage permet un typage statique optionnel des variables et des fonctions, la création de classes et d'interfaces, l'import de modules. Il fournit une compatibilité avec ECMAScript 6. Cette norme permet l'ajout de classes et de modules notamment.

Comme nous l'avons vu, la modularité est une part importante de ce prototype. La validation statique du code nous permet d'avoir une meilleure production sur les types fournit ainsi que sur les modules.

Pour être utilisé, le code source doit du coup être compilé, d'où la nécessité d'avoir une chaîne de construction, ce que nous verrons dans la section 7.3. L'intérêt de Typescript est directement lié à l'utilisation d'angular et de son routage.

7.2 AngularJS et UI-Router

Nous ne présenterons pas de détail sur la technologie AngularJS ici. Nous n'allons qu'explorer son intérêt pour notre application.

AngularJS est un *framework* MVC (Modèle, Vue, Contrôleur) pour les applications web. Ses principes de conceptions sont les suivants :

- Séparation de la vue ainsi que sa manipulation (i.e. du DOM) et la logique métier.
- L'injection de dépendance entre composants.
- Une gestion de cycle de vie par l'utilisation de *promises*.

L'injection de dépendance est une partie centrale d'Angular qui permet le découplage des services qui permettent notamment de dialoguer avec l'API Backend et la logique du contrôleur.

Par exemple : voici le code d'un service simple en Angular/Typescript

```
import IHttpPromise = angular.IHttpPromise;
export class SampleService {
    private http:ng.IHttpService;
    private log:ng.ILogService;

    constructor($http, $log) {
        this.http = $http;
        this.log = $log;
    }

    public getData():IHttpPromise<MyData> {
        this.log.debug("Getting data from server");
        return this.http.get('/api/data');
    }
}
```

Ce service a une dépendance vers $http (qui est le service permettant de faire des appels http au serveur) et $log. Chaque attribut utilisé est typé et sera statiquement validé par un compilateur. Ce service expose la méthode getData qui fournit un objet de type SampleData qui sera obtenu par une promesse http elle aussi statiquement typé.

Ce service pourra être utilisé à tout endroit dans le composant qui l'a déclaré.

Cela est couplé avec UI-Router qui permet de définir la navigation dans l'application par une arborescence d'état. Ci-dessous, la déclaration d'un état

```
$stateProvider.state('product.sample', {
    url: '/product/sample',
    views: {
        "@": {
            template: <string>require('./sample.html'),
            controller: 'SampleController',
            controllerAs: 'Sample',
            resolve: {
```

```
                sampleData: (SampleService) => {
                    return SampleService.getData();
                }
            }
        }
    }
});
```

Cet état est un état fils de *product*, qui est atteignable en allant à la page /product/sample. Une fois cet état activé, le routeur passera par une étape de résolution où pour chaque vue déclarée, les promesses seront complétées. La vue ne sera jamais active tant que les promesses n'auront pas été résolues. La donnée ainsi résolue (ici *sampleData*) sera disponible par injection au contrôleur.

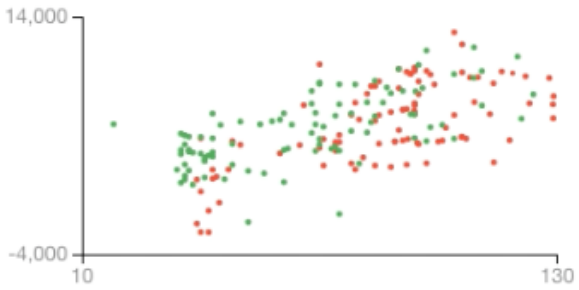
Enfin, en utilisant les mêmes principes, Angular permet la création de composants (appelées *directives*) qui peuvent s'utiliser comme des balises html étendues et qui, comme dans tout modèle à composants, peuvent dépendre d'autres composants.

Par exemple, il est possible de constituer une directive permettant de tracer un graphique

```
<plot chart-data="Sample.sampleData" new-entry="Sample.newEntry"></plot>
```

La directive *plot* possède deux paramètres qui sont des objets disponibles dans le modèle du contrôleur (appelé le *scope*) et permet l'affichage d'un nuage de point par exemple. En l'occurrence ici, le premier est le résultat de la promesse que nous avions montré précédemment, la mention de « Sample. » permet de pointer vers le bon contrôleur.

Figure 5 - Affichage de la directive plot



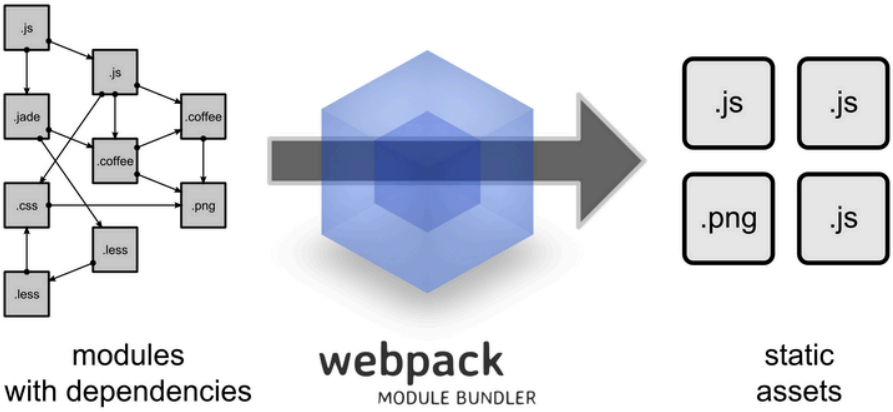
Les composants sont des produits à part entière puisqu'ils peuvent posséder leurs propres services et leurs propres sous-composants. Ainsi, nous pouvons constituer une bibliothèque de *directives*. Cette directive « plot » a été d'ailleurs utilisée pour le premier démonstrateur Waves.

7.3 Webpack

Webpack est une chaîne de construction qui permet d'opérer l'assemblage des différentes sources en une application web finale. Il prend en entrée un ensemble de fichiers ayant des

dépendances entre eux (avec tout types de langages) et fournira des fichiers interprétables par un navigateur en sortie.

Figure 6 - Webpack



Son intérêt est évident dans notre cas. Toutefois, la jeunesse de ces types de produit nous montre qu'il est difficile de trouver une configuration de webpack permettant l'utilisation des technologies que nous avons présenté jusqu'ici.

Nous avons donc dans le cadre de ce projet constitué un projet principal permettant d'avoir la configuration nécessaire pour utiliser Typescript, Angular et Webpack. Le code et sa documentation sont décrit ici https://github.com/deonclem/angular-webpack-typescript

7.4 D3.js

D3.js est une bibliothèque graphique Javascript permettant la visualisation de données sous forme graphique et dynamique. Il se base sur les technologies SVG et CSS pour permettre un rendu vectoriel de la donnée.

Son approche est orientée donnée, c'est à dire que la visualisation est construite en parcourant les données. Ici, un exemple extrait de la directive *plot*.

```
let dots = svg.data(chartData, (d:any) => d.timestamp)
                .enter()
                .append("circle")
                .attr("class", "dot")
                .attr("r", "2")
                .attr("cx", (d:any) => d.x)
                .attr("cy", (d:any) => d.y);
```

Ce code peut être lu ainsi : « avec les données issues de chartData, pour chaque point, créer un rond de classe css « .dot » de rayon 2 aux coordonnées correspondant aux attributs x et y ». Cet aspect à la limite du déclaratif permet de plus facilement constituer des visualisations interactives.

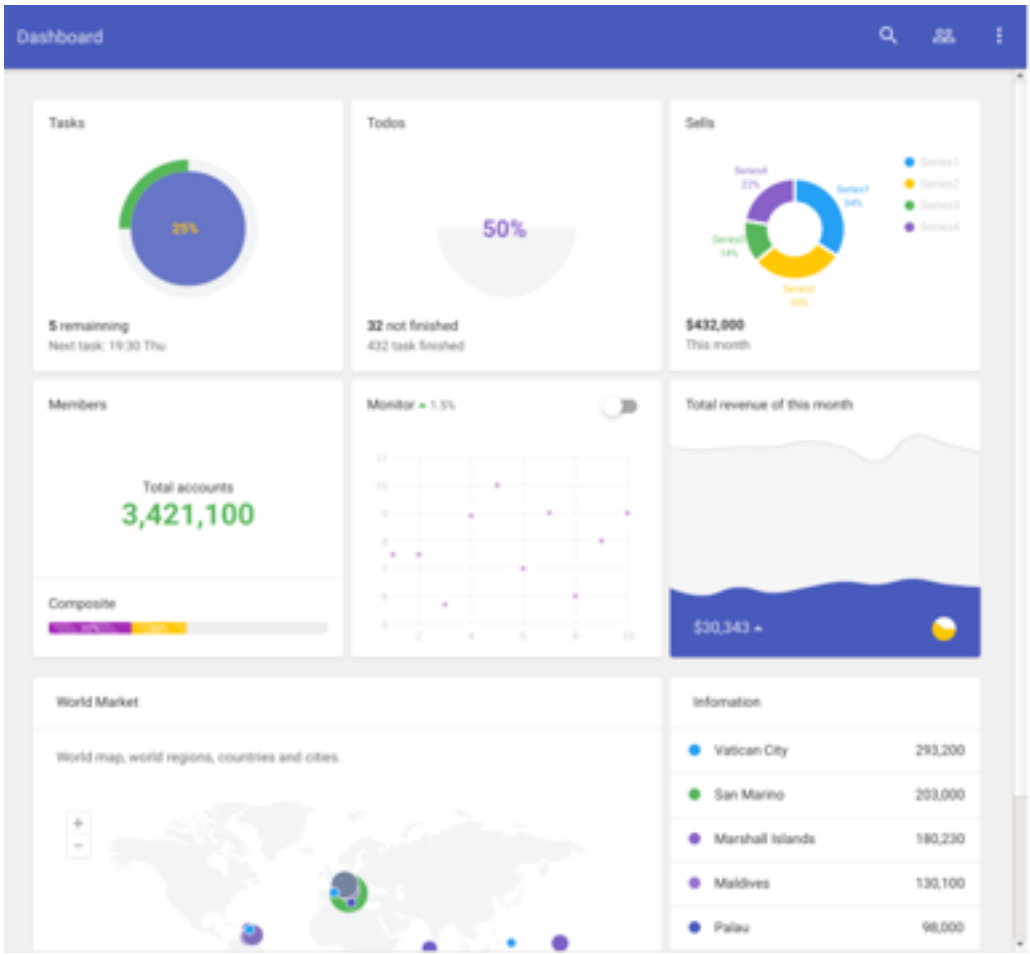
7.5 Angular-Material

Enfin, concernant la présentation de l'application et l'assemblage des composants. Nous utilisons Angular-Material. Cette bibliothèque est l'implémentation de référence de la spécification *Google's Material Design*. Cette spécification fournit système unifié de design et d'interaction d'applications. Ce système a l'avantage de s'adapter à tout support et tailles d'écrans.

En pratique, cette bibliothèque fournit des classes CSS et des directives angular permettant d'assembler une application web de façon plus efficace tout en restant agréable à la vue et à la navigation.

Dans les directives fournies, nous pouvons retrouver des primitives de mises en page, ou des fenêtres de dialogues ou encore des formulaires d'autocomplétion. Au fur et à mesure du temps, beaucoup de bibliothèques ont fourni ce type de fonctionnalité, mais celle-ci, en dehors de son aspect visuellement plus clair, est parfaitement intégrée dans la technologie Angular (contrairement à Bootstrap par exemple). Ceci rendant son utilisation plus simple en permettant notamment de régler tout ce qui concerne la vue dans le html via des directives.

Figure 7 - Exemple de tableau de bord en material



8 Conclusion

Nous avons ici présenté les premières maquettes de notre application Waves. Et nous avons décrit la pile technique que nous utilisons pour y parvenir. Nous avons développé cet assemblage afin d'obtenir la plus grande modularité en terme de développement et d'utilisation.

Lors de l'implémentation du premier démonstrateur. L'entièreté de cette pile technique a été utilisée dans la forme qui a été présentée. La suite sera donc sur le fait de rendre l'application extensible par rapport aux données disponibles dans le *framework* Waves.