



Il était une table

DOCUMENTATION TECHNIQUE & FONCTIONNELLE



| | |
|---|-----------|
| I. Le projet | 3 |
| A. Cadre du projet | 3 |
| B. Charte graphique | 3 |
| C. Liste des fonctionnalités | 4 |
| II. Description fonctionnelle | 5 |
| A. Principe fonctionnel | 5 |
| B. Structuration de la page | 5 |
| <i>B.1. Header</i> | 6 |
| <i>B.2. Filtre</i> | 7 |
| <i>B.3. Carte</i> | 8 |
| <i>B.4. Liste des restaurants</i> | 10 |
| <i>B.5. Fenêtre « Avis de restaurant »</i> | 12 |
| <i>B.6. Fenêtre « Photo »</i> | 14 |
| <i>B.7. Fenêtre « Ajout d'un avis »</i> | 15 |
| <i>B.8. Fenêtre « Comment ajouter un restaurant »</i> | 17 |
| <i>B.9. Fenêtre « Ajouter un restaurant »</i> | 19 |
| III. Description technique | 21 |
| A. Choix technologiques | 21 |
| B. Architecture | 22 |
| C. Diagramme de classes | 24 |
| D. Diagrammes de séquence | 25 |
| <i>D.1. Fonctionnalités</i> | 25 |
| <i>D.2. Fragments</i> | 34 |
| IV. Réflexion autour du projet : limites et opportunités d'évolution | 39 |

I. Le projet

A. Cadre du projet

Objectif :

Créer un service simple et utile qui permette à l'utilisateur d'obtenir des avis sur les restaurants qui se trouvent autour de lui.

Besoins / Contraintes :

- Développer l'application en Javascript
- Exploiter les APIs de Google

B. Charte graphique

Principe :

Le design de l'application s'inspire des principes du Material Design.

Police de caractères :

Logo : « Sacramento »

- *Il était une table*

Titres et textes : « Roboto Condensed »

- Texte normal
- **Texte en gras**
- TEXTE EN MAJUSCULE LÉGER

Palettes de couleurs :



C. Liste des fonctionnalités

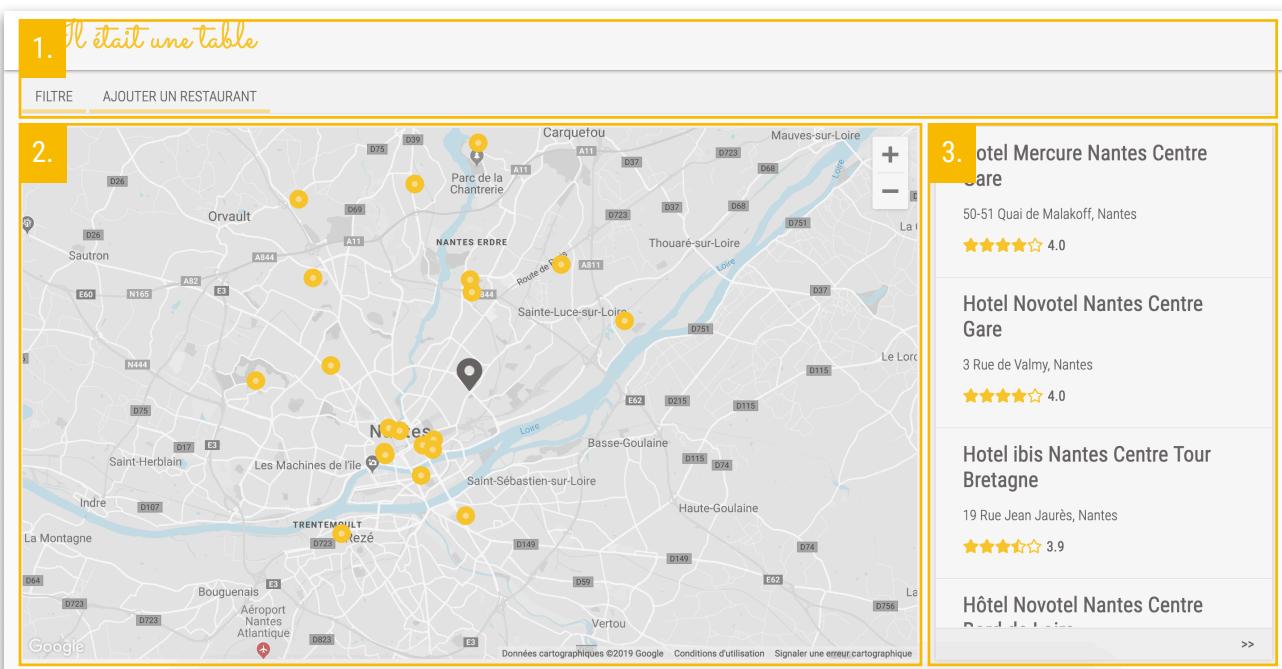
| Front-end |
|---|
| Afficher une carte google |
| Centrer la carte sur la position de l'utilisateur |
| Afficher les restaurants sur la carte |
| Afficher les restaurants visibles sur la carte sous forme de liste à côté de la carte |
| Filtrer les restaurants en fonction de leur note moyenne |
| Afficher la note moyenne d'un restaurant |
| Afficher le détail des avis des restaurants |
| Afficher la photo Google Street View d'un restaurant |
| Ajouter un nouvel avis à un restaurant existant |
| Ajouter un nouveau restaurant en cliquant sur la carte |
| Back-end |
| Obtenir la position de l'utilisateur sous réserve de son accord préalable via l'API Geolocation |
| Obtenir une carte Google via l'API Maps Javascript |
| Obtenir des restaurants dans une zone affichée via l'API Google Places |
| Obtenir des avis pour les restaurants retournés par l'API Google Places |
| Obtenir l'adresse d'un lieu à partir de ses coordonnées grâce à l'API Geocoding |
| Obtenir la photo Google Street View d'un restaurant via l'API Street View Static |
| Enregistrer des restaurants dans le stockage local du navigateur |
| Modifier des propriétés de restaurants enregistrés dans le stockage local du navigateur |
| Assurer la persistance des modifications apportées à un ou plusieurs restaurants |

II. Description fonctionnelle

A. Principe fonctionnel

L'interface de l'application comprend une unique page. Elle s'articule autour d'un header et de deux sections principales dépendantes : une carte, à gauche, où s'affichent les marqueurs des restaurants et une liste à droite qui répertorient les restaurants qui apparaissent sur la carte. La hauteur de l'interface est celle du viewport, ainsi la liste des restaurants est toujours à hauteur de la carte. Les restaurants sont affichés par paquet de 20 pour que la carte reste lisible et la liste propose un système de pagination. Les informations complémentaires, telles que le détail des avis, la photo Google Street View, l'ajout d'un nouveau restaurant, l'ajout d'un nouvel avis, sont affichées dans des fenêtres modales.

B. Structuration de la page



B.1. Header

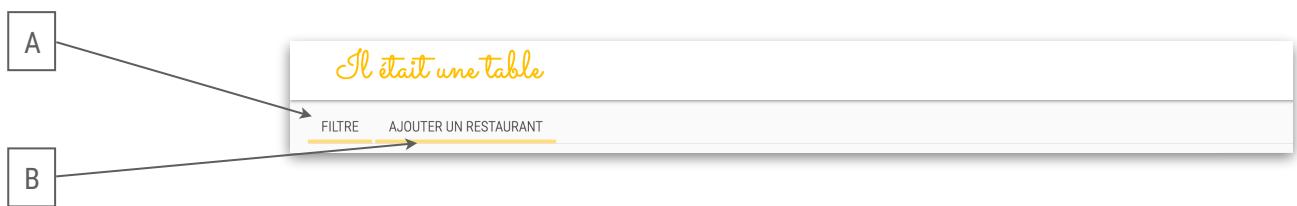


Figure 2 : Wireframe « Header »

Objectif :

Le header affiche le nom du site et offre des options à l'utilisateur pour utiliser l'application.

Composition :

Le header se compose :

- du nom du site
- d'une barre de menus.

Interactions du point de vue d'un utilisateur :

L'utilisateur peut interagir avec la barre de menus. Il peut cliquer sur « Filtre » ou sur « Ajouter un restaurant ». Au survol des menus, les titres des menus changent d'apparence.

A. Quand il clique sur « Filtre », le filtre par note moyenne s'affiche.

B. Quand il clique sur « Ajouter un restaurant », une fenêtre l'informant de la manière d'ajouter un nouveau restaurant s'affiche.

| Bloc | Contenu | Caractéristiques | Commentaires |
|----------------|---------|--|--|
| Nom du site | Texte | - <h1> - font-family : « Sacramento » - color : #FFC107 | |
| Barre de menus | Menus | - <div> | |
| Menus | Texte | - majuscule - border-bottom - couleur de la bordure : #FFE082. | Au survol, les menus changent d'apparence. La graisse de la police est plus épaisse. La bordure qui souligne le titre change de couleur (#FFC107). |

B.2. Filtre



Figure 3 : Wireframe « Filtre »

Objectif :

Le filtre apparaît après que l'utilisateur ait cliqué sur « FILTRE ». Il permet à l'utilisateur de filtrer les restaurants en fonction de leur note moyenne, en choisissant un intervalle de notes.

Composition :

Le filtre se compose :

- d'un titre,
- d'un double range slider,
- de labels indiquant les notes minimum et maximum sélectionnées.

Interactions du point de vue d'un utilisateur :

A. Pour afficher le filtre, l'utilisateur doit cliquer sur le menu « Filtre ». Il ne s'affiche plus quand la souris ne survole plus la div contenant le filtre.

B. Pour modifier l'intervalle de notes, l'utilisateur fait glisser les poignées du slider. Seuls les labels des notes sélectionnées s'affichent. L'ajustement des restaurants qui apparaissent sur la carte, et par conséquent dans la liste, se fait instantanément.

| Bloc | Contenu | Caractéristiques | Commentaires |
|--------|-----------|-----------------------|---|
| Titre | Texte | - <p> | |
| Slider | Forme CSS | - double range slider | Le slider est généré à l'aide de la librairie JS noUiSlider et personnalisé à l'aide de quelques règles CSS pour s'accorder avec la charte graphique. |
| Notes | Texte | - | Seules les notes actives sont visibles. |

B.3. Carte

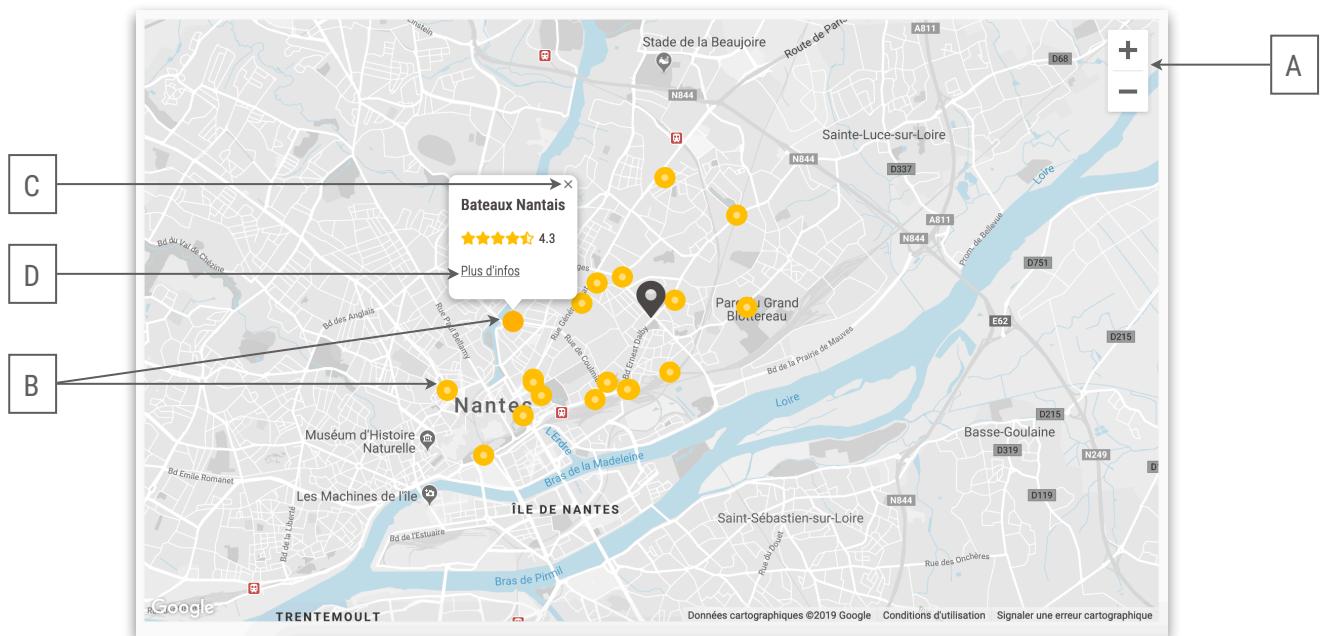


Figure 4 : Wireframe « Carte »

Objectif :

La carte affiche les marqueurs des restaurants qui se trouvent autour de l'utilisateur, dont la position est signifiée par un marqueur qui lui est propre.

Composition :

La carte se compose

- d'une carte google,
- du marqueur de l'utilisateur,
- des marqueurs des restaurants,
- des *infowindows* liées aux marqueurs des restaurants.

Interactions du point de vue d'un utilisateur :

A. L'utilisateur peut se déplacer sur la carte en la faisant glisser à l'aide du curseur. Il peut aussi zoomer et dézoomer pour changer d'échelle, à l'aide du curseur ou des boutons dédiés.

B. Au survol sur les marqueurs des restaurants, ils changent d'apparence.

C. Au clic sur un marqueur de restaurant, l'*infowindow* correspondante s'ouvre. Elle se ferme en cliquant sur la croix en haut à droite, ou si l'utilisateur clique sur un autre marqueur.

D. Pour obtenir davantage d'informations sur le restaurant, l'utilisateur peut cliquer sur la mention « Plus d'infos » d'une *infowindow*.

| Bloc | Contenu | Caractéristiques | Commentaires |
|----------------------|------------------------|--|---|
| Carte | Google Maps | <ul style="list-style-type: none"> - La carte est contenue dans une <div> dont la hauteur est calculée de manière à ce que la hauteur de l'interface corresponde toujours à la hauteur du viewport. | La carte est générée à l'aide de l'API Google : Maps Javascript API. Le style de la carte est contenu dans un fichier JSON. |
| Marqueur utilisateur | Marker Google Maps | <ul style="list-style-type: none"> - La forme du marqueur est personnalisée, c'est une forme SVG. - Couleur : #4E4946 | Le marqueur est positionné grâce à l'API W3C Geolocation qui retourne la position de l'utilisateur, si celui-ci a donné son accord et si le navigateur qu'il utilise supporte le service. |
| Marqueur restaurant | Marker Google Maps | <ul style="list-style-type: none"> - La forme du marqueur est personnalisée. Il s'agit d'une forme SVG qui change au survol de la souris sur le marqueur. - Couleurs icône classique: #FFC107, #FFE082 - Couleur icône survolée : #FFB300 | Le marqueur est positionné à l'aide des coordonnées géographiques du restaurant. |
| Infowindow | InfoWindow Google Maps | <ul style="list-style-type: none"> - <h3> pour le nom du restaurant - Les étoiles sont des icônes Font Awesome - <a> pour le lien vers les avis. | Une seule infowindow peut être affichée à la fois. |

B.4. Liste des restaurants

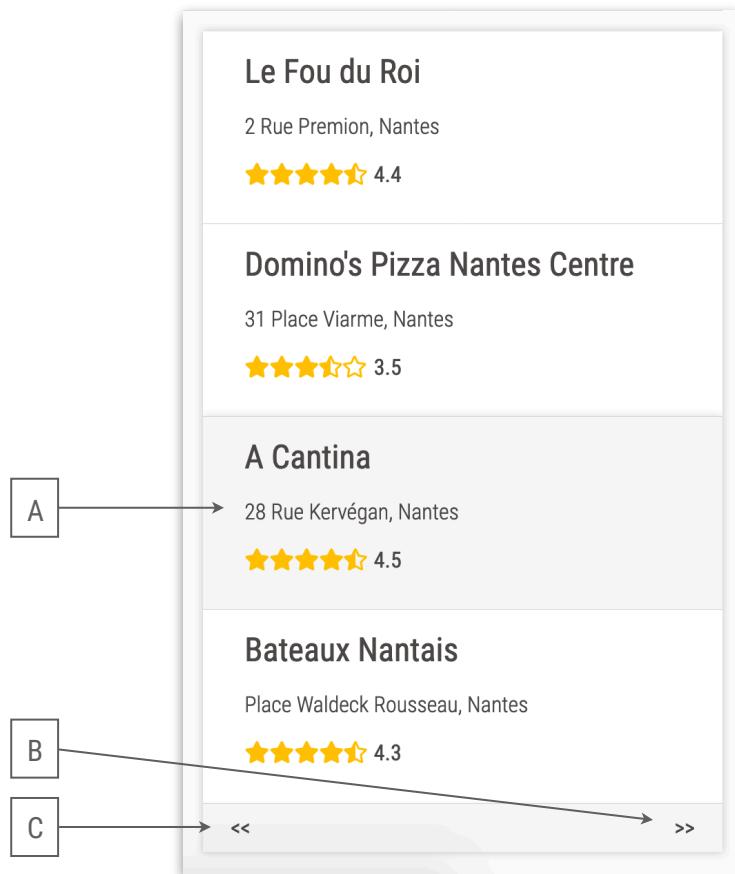


Figure 5 : Wireframe « Liste des restaurants »

Objectif :

La liste affiche le résumé des restaurants qui s'affichent sur la carte. La liste comporte au maximum 20 résultats. S'il y a plus de 20 restaurants à afficher dans la liste, une barre de pagination permet de naviguer entre les résultats.

Composition :

La liste se compose :

- des cards des restaurants affichés sur la carte,
- d'une barre de pagination comprenant un bouton pour afficher les résultats suivants et un bouton pour afficher les résultats précédents.

Interactions du point de vue d'un utilisateur :

- Au survol d'une card d'un restaurant, la card change d'apparence. Et au clic sur la card, une fenêtre contenant les avis reçus par le restaurant s'affiche.
- Si dans la bande en bas de la liste, il y a un bouton « >> », au clic sur celui-ci, la liste affiche les résultats suivants.
- De la même manière, s'il y a un bouton « << », au clic sur celui-ci, la liste affiche les 20 résultats précédents.

| Bloc | Contenu | Caractéristiques | Commentaires |
|---------------------------------|-----------------------|---|--|
| Liste | Cards | <ul style="list-style-type: none"> - <div> - Overflow : scroll | La hauteur de la liste est la même que la hauteur de la carte. |
| Card | Texte | <ul style="list-style-type: none"> - <h2> pour le nom du restaurant - <p> pour l'adresse - Les étoiles sont des icônes Font Awesome - Evénement « click » sur l'élément <a> qui recouvre la card pour accéder aux avis du restaurant. | La card d'un restaurant change d'apparence quand l'utilisateur la survole. |
| Barre de pagination | Boutons de pagination | <ul style="list-style-type: none"> - <div> - position : absolute | |
| Bouton « résultats précédents » | Texte | <ul style="list-style-type: none"> - <div> - Caractères : << | Il s'affiche uniquement s'il y a une page précédente à afficher. |
| Bouton « résultats suivants » | Texte | <ul style="list-style-type: none"> - <div> - Caractères : >> | Il s'affiche uniquement s'il y a une page suivante à afficher. |

B.5. Fenêtre « Avis de restaurant »

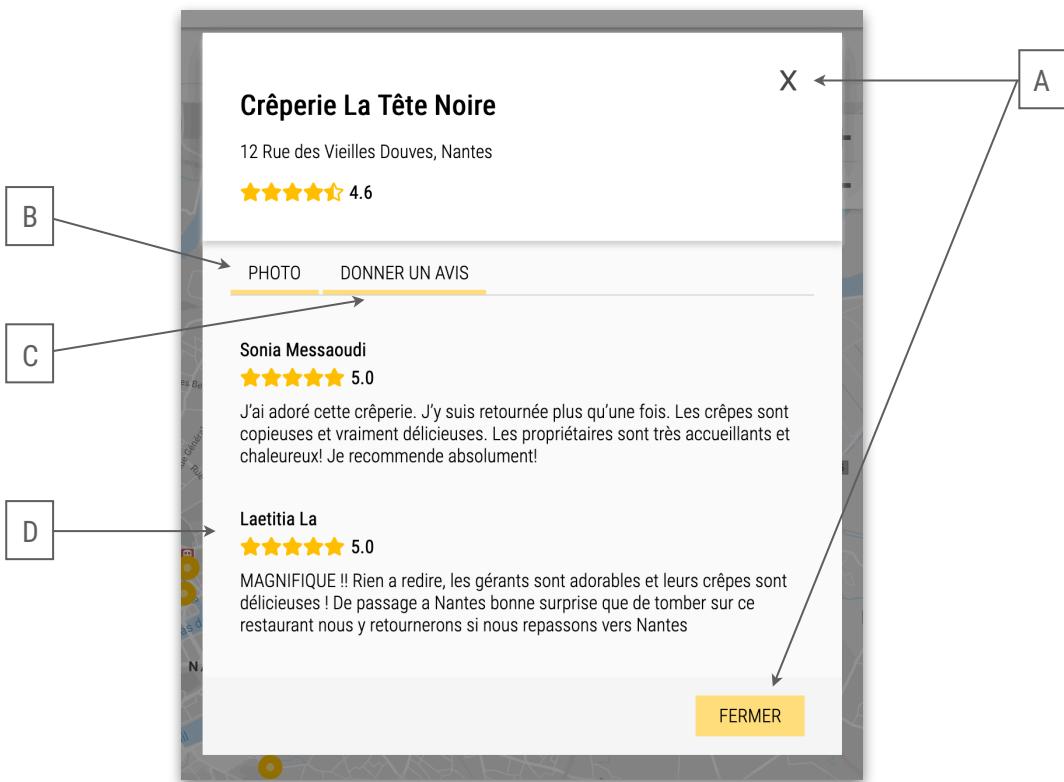


Figure 6 : Wireframe « Fenêtre : avis de restaurant »

Objectif :

Cette fenêtre affiche les critiques reçues par le restaurant

Composition :

La fenêtre se compose :

- d'un en-tête comprenant les informations relatives au restaurant : nom, adresse, note moyenne
- d'une croix pour fermer,
- une barre de menus pour afficher la photo du restaurant, si elle existe, et pour ajouter un avis
- la liste des avis (chaque avis précise le nom de l'utilisateur, la note attribuée et le commentaire laissé)
- un footer comprenant un bouton pour fermer la fenêtre.

Interactions du point de vue d'un utilisateur :

A. L'utilisateur peut fermer la fenêtre en cliquant sur le bouton « Fermer » dans le footer ou en cliquant sur la croix en haut dans l'en-tête. Au survol, ces boutons changent d'apparence.

B. L'utilisateur peut afficher la photo du restaurant en cliquant sur « photo ». La photo sera affichée dans une nouvelle fenêtre modale. Si aucune photo n'est disponible, le menu est grisé. Au survol, le titre du menu change d'apparence.

C. L'utilisateur peut ajouter un avis en cliquant sur « donner un avis ». Une nouvelle fenêtre modale s'ouvre alors. Au survol, le titre du menu change d'apparence.

D. Si le restaurant a reçu de nombreux avis, l'utilisateur doit scroller pour les consulter.

| Bloc | Contenu | Caractéristiques | Commentaires |
|-------------------|---------|---|---|
| Fenêtre modale | | | La fenêtre est générée à l'aide de la librairie javascript Alertify. |
| En-tête | Texte | <ul style="list-style-type: none"> - <h2> : nom du restaurant - <p> : adresse - Les étoiles sont des icônes Font Awesome | |
| Croix | Texte | <ul style="list-style-type: none"> - Pseudo-élément CSS | La croix permet la fermeture de la fenêtre modale. |
| Barre de menus | Menus | <ul style="list-style-type: none"> - <div> | Les menus permettent d'accéder à la photo Street View et d'ajouter un avis. |
| Menu | Texte | <ul style="list-style-type: none"> - majuscule - border-bottom - couleur de la bordure : #FFE082. | Au survol, les menus changent d'apparence. La graisse de la police est plus épaisse. La bordure qui souligne le titre change de couleur (#FFC107). Si la photo Street View n'est pas disponible, le menu est grisé. |
| Liste des avis | Avis | <ul style="list-style-type: none"> - <div> - overflow : scroll | |
| Avis | Texte | <ul style="list-style-type: none"> - <p> - Les étoiles sont des icônes Font Awesome. | |
| Footer | Bouton | <ul style="list-style-type: none"> - <div> | |
| Bouton « Fermer » | Texte | <ul style="list-style-type: none"> - Majuscule - background-color : #FFE082 | Au survol et au clic sur le bouton, l'arrière plan change de couleur : #FFC107. |

B.6. Fenêtre « Photo »

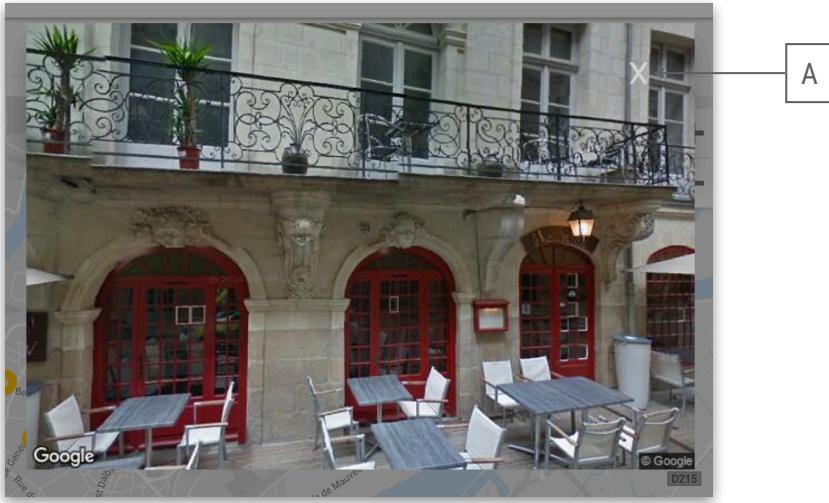


Figure 7 : Wireframe « Fenêtre : photo »

Objectif :

La fenêtre affiche la photo Google Street View du restaurant.

Composition :

La fenêtre contient :

- la photo du restaurant,
- une croix pour fermer la fenêtre.

Interactions du point de vue d'un utilisateur :

A. Pour fermer la fenêtre, l'utilisateur peut cliquer sur la croix en haut à droite de la fenêtre.

| Bloc | Contenu | Caractéristiques | Commentaires |
|--------------------|---------|--|---|
| Fenêtre modale | | | La fenêtre est générée à l'aide de la librairie javascript Alertify. |
| Photo | Image | <ul style="list-style-type: none">- - href : lien vers la photo google street view du restaurant- alt : nom du restaurant | |
| Croix de fermeture | Texte | - Pseudo-élément CSS | La croix permet la fermeture de la fenêtre et de revenir à la fenêtre des avis. |

B.7. Fenêtre « Ajout d'un avis »

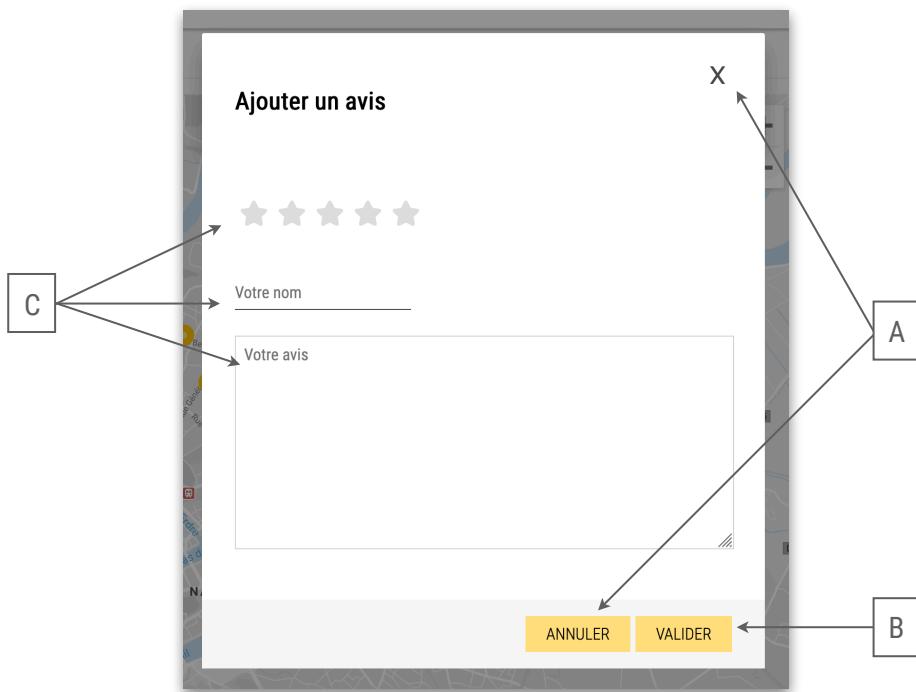


Figure 8 : Wireframe « Fenêtre : ajout d'un avis »

Objectif :

La fenêtre permet à l'utilisateur d'ajouter un avis.

Composition :

La fenêtre se compose :

- d'un titre,
- d'une croix de fermeture,
- d'un formulaire comprenant :
 - d'une série d'étoiles pour noter le restaurant,
 - d'un champ de saisie du nom de l'utilisateur,
 - d'une zone de texte pour saisir le commentaire,
- d'un footer avec un bouton d'annulation et un bouton de validation.

Interactions du point de vue d'un utilisateur :

A. Pour quitter la fenêtre sans enregistrer l'avis, l'utilisateur peut cliquer sur la croix ou sur le bouton « Annuler ».

B. Pour enregistrer l'avis, l'utilisateur doit cliquer sur le bouton « Valider ». Au préalable, il doit avoir rempli les différents champs : note, nom et commentaire.

Au survol, ces trois boutons changent d'apparence.

C. L'utilisateur peut remplir les trois champs : note, nom et avis.

| Bloc | Contenu | Caractéristiques | Commentaires |
|--------------------|------------------|---|---|
| Fenêtre modale | | | La fenêtre est générée à l'aide de la librairie javascript Alertify. |
| Titre | Texte | - <h2> | |
| Croix de fermeture | Texte | - Pseudo-élément CSS | La croix ferme la fenêtre sans enregistrer l'avis. Après la fermeture de la fenêtre, l'utilisateur retourne à la fenêtre des avis. |
| Formulaire | Champs de saisie | - <form> | Tous les champs du formulaire doivent être saisis pour pouvoir valider l'envoi. |
| Etoiles | Champ de saisie | - <input> et <label> - Les étoiles sont des icônes Font Awesome. | |
| Nom | Champ de saisie | - <input> | |
| Commentaire | Zone de texte | - <textarea> | |
| Footer | Boutons | - <div> | |
| Bouton « Annuler » | Texte | - Majuscule - background-color : #FFE082 | Au survol et au clic sur le bouton, l'arrière plan change de couleur : #FFC107. Au clic sur le bouton « Annuler », la fenêtre se ferme sans enregistrer d'avis et retourne à la fenêtre précédente des avis. |
| Bouton « Valider » | Texte | - Majuscule - background-color : #FFE082 | Au survol et au clic sur le bouton, l'arrière plan change de couleur : #FFC107. Au clic sur le bouton « Valider », si tous les champs sont remplis, la fenêtre se ferme et l'utilisateur retourne à la fenêtre des avis où l'avis ajouté tout récemment apparaît désormais dans la liste et où la note moyenne a été mise à jour en conséquence. |

B.8. Fenêtre « Comment ajouter un restaurant »

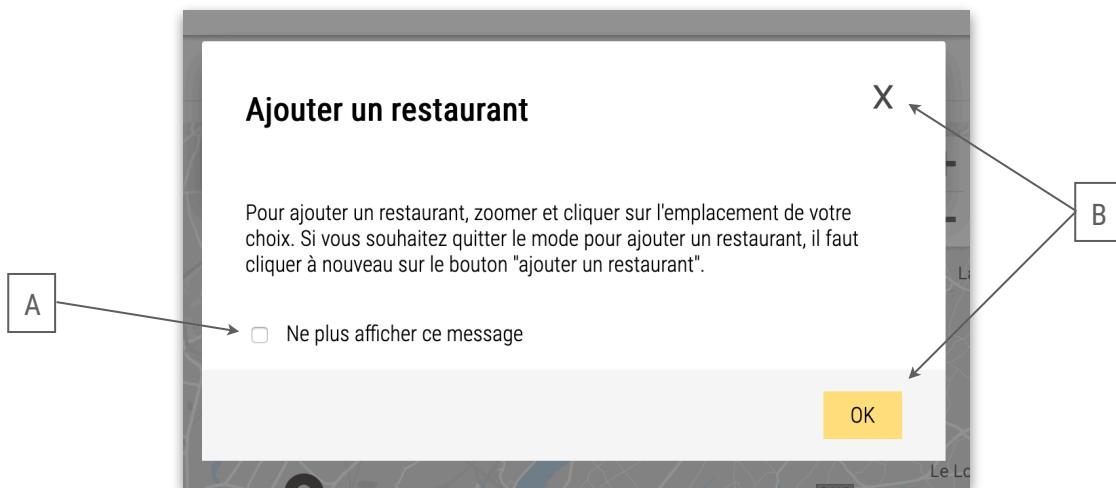


Figure 9 : Wireframe « Fenêtre : comment ajouter un restaurant »

Objectif :

La fenêtre informe sur la manière d'ajouter un restaurant.

Composition :

La fenêtre se compose :

- d'un titre,
- d'une croix de fermeture,
- d'un texte informatif
- d'un formulaire pour ne plus afficher le message,
- d'un footer comprenant un bouton « ok ».

Interactions du point de vue d'un utilisateur :

A. L'utilisateur peut cocher la case pour que la fenêtre ne s'affiche plus les fois suivantes où il cliquera sur le menu « Ajouter un restaurant ». Cette mention s'affiche uniquement si l'utilisateur a accepté les cookies.

B. L'utilisateur peut fermer la fenêtre en cliquant sur la croix ou sur le bouton « OK », qui changent d'apparence quand l'utilisateur les survole.

| Bloc | Contenu | Caractéristiques | Commentaires |
|--------------------|-----------------|---|---|
| Fenêtre modale | | | La fenêtre est générée à l'aide de la librairie javascript Alertify. |
| Titre | Texte | - <h2> | |
| Croix de fermeture | Texte | - Pseudo-élément CSS | La croix ferme la fenêtre et permet à l'utilisateur d'entrer dans le mode d'ajout de restaurant. Le curseur au survol de la carte est alors différent. |
| Texte informatif | Texte | - <p> | |
| Formulaire | Champ de saisie | - <input> de type checkbox - <label> | Si l'utilisateur coche la case et appuie sur le bouton « OK » dans le footer, un cookie est enregistré et la fenêtre ne s'affiche plus après avoir cliqué sur le menu « ajouter un restaurant ». |
| Footer | Boutons | - <div> | |
| Bouton «OK » | Texte | - Majuscule - background-color : #FFE082 | Au survol du bouton, le couleur du bouton change : #FFC107. Au clic sur le bouton, la fenêtre se ferme. Si la case pour ne plus afficher la fenêtre est cochée, un cookie est enregistré. Le curseur change d'apparence au survol de la carte et signifie que l'utilisateur est dans le mode pour ajouter un restaurant. |

B.9. Fenêtre « Ajouter un restaurant »

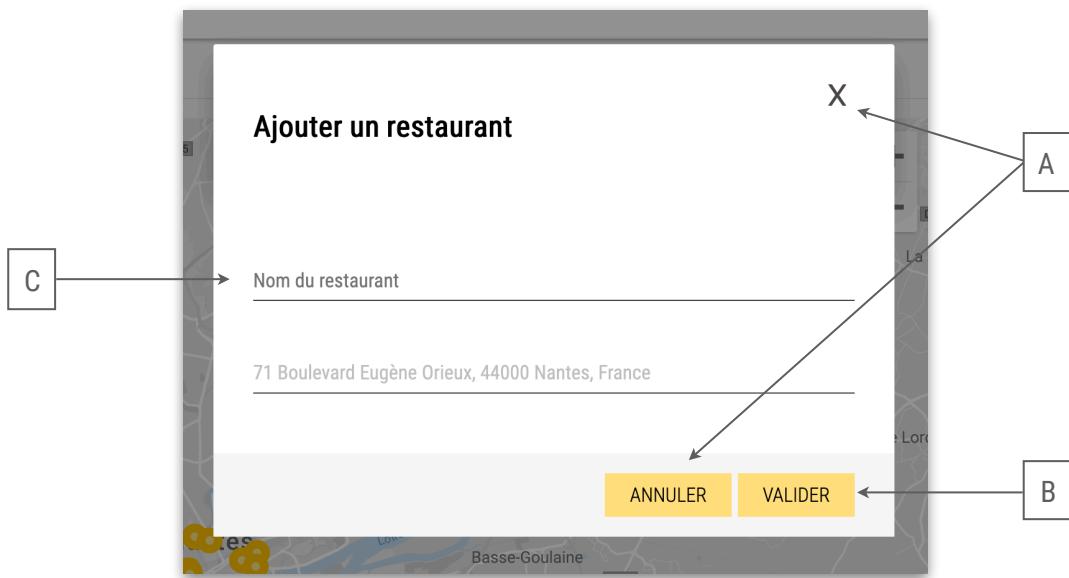


Figure 10 : Wireframe « Fenêtre : ajouter un restaurant »

Objectif :

La fenêtre permet d'ajouter un restaurant après que l'utilisateur ait cliqué sur un point de la carte et qui renvoyait une adresse.

Composition :

La fenêtre se compose :

- d'un titre,
- d'une croix de fermeture,
- d'un formulaire :
 - un champ de saisie du nom du restaurant,
 - un champ affichant l'adresse du restaurant,
- d'un footer comprenant un bouton « annuler » et un bouton « valider ».

Interactions du point de vue d'un utilisateur :

A. Pour quitter la fenêtre sans enregistrer le restaurant, l'utilisateur peut cliquer sur la croix ou sur le bouton « Annuler », qui changent d'apparence quand ils sont survolés.

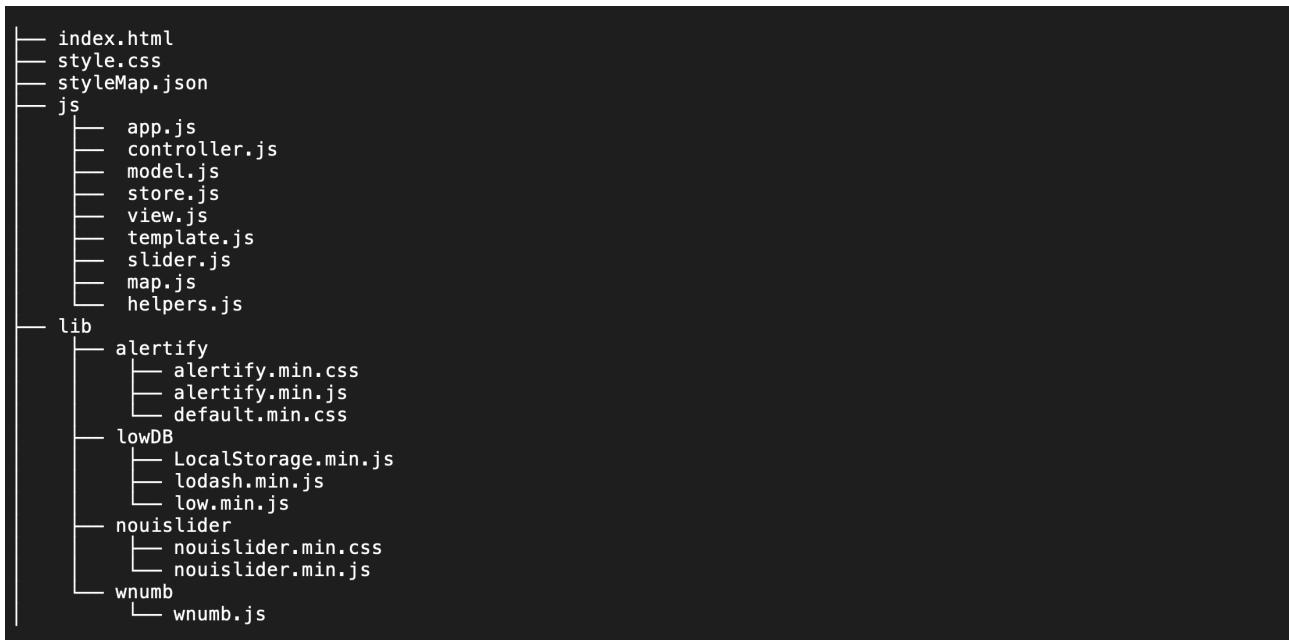
B. Pour enregistrer le nouveau restaurant, l'utilisateur doit cliquer sur le bouton « Valider », qui change d'apparence quand il est survolé. Au préalable, il doit avoir renseigné le champ « Nom du restaurant ». Le champ contenant l'adresse du restaurant n'est pas modifiable.

C. L'utilisateur peut compléter le champ « Nom du restaurant ».

| Bloc | Contenu | Caractéristiques | Commentaires |
|-----------------------|------------------|--|--|
| Fenêtre modale | | | La fenêtre est générée à l'aide de la librairie javascript Alertify. |
| Titre | Texte | - <h2> | |
| Croix de fermeture | Texte | - Pseudo-élément CSS | La croix ferme la fenêtre sans enregistrer le nouveau restaurant et quitte le mode d'ajout de restaurant. Le curseur au survol de la carte retrouve son aspect initial. |
| Formulaire | Champs de saisie | - <form> | |
| Nom du restaurant | Champ de saisie | - <input> - placeholder : Nom du restaurant | Ce champ doit être renseigné pour valider l'ajout d'un restaurant. |
| Adresse du restaurant | Champ de saisie | - <input> - value : adresse du lieu cliqué | Ce champ n'est pas modifiable. L'adresse dans le champ est retournée grâce à l'API Geocoding. |
| Footer | Boutons | - <div> | |
| Bouton « Annuler » | Texte | - Majuscule - background-color : #FFE082 - Au survol et au clic sur le bouton, l'arrière plan change de couleur : #FFC107. | Au clic sur le bouton « Annuler », la fenêtre se ferme sans enregistrer le nouveau restaurant. L'utilisateur quitte le mode d'ajout de restaurant et le curseur au survol de la carte retrouve son aspect initial. |
| Bouton « Valider » | Texte | - Majuscule - background-color : #FFE082 - Au survol et au clic sur le bouton, l'arrière plan change de couleur : #FFC107. | Au clic sur le bouton « Valider », si le nom du restaurant est saisi, la fenêtre se ferme, le nouveau restaurant apparaît sur la carte et dans la liste. L'utilisateur quitte le mode d'ajout de restaurant et le curseur au survol de la carte retrouve son aspect initial. |

III. Description technique

A. Choix technologiques



```
index.html
style.css
styleMap.json
js
  app.js
  controller.js
  model.js
  store.js
  view.js
  template.js
  slider.js
  map.js
  helpers.js
lib
  alertify
    alertify.min.css
    alertify.min.js
    default.min.css
  lowDB
    LocalStorage.min.js
    lodash.min.js
    low.min.js
  nouislider
    nouislider.min.css
    nouislider.min.js
  wnumb
    wnumb.js
```

Figure 10 : Arborescence des fichiers de l'application

L'application fonctionne à l'aide d'un ensemble de fichiers javascript, écrits en Vanilla JS. Elle s'appuie sur plusieurs librairies JS et fait appel à plusieurs API externes. Elle intègre également des CDN pour accéder aux icônes et aux polices de caractères.

Librairies JS

- **noUiSlider** permet de générer un *double range slider*. Le slider a pour objectif de filtrer les restaurants en indiquant un intervalle de notes moyennes.
- **wNumb** est un outil complémentaire de noUiSlider. Il permet de personnaliser le format décimal des notes moyennes sur le slider.
- **Alertify** permet de générer des fenêtres modales.
- **lowDB** est une base de données locales. Elle va permettre de stocker les restaurants retournés par les requêtes lancées via les API de Google et ceux ajoutés par l'utilisateur lui-même.

API Externes

- **Maps Javascript API** va permettre d'afficher une carte dynamique et d'y placer des marqueurs grâce aux coordonnées géographiques des restaurants et de l'utilisateur.
- **StreetView Static API** va permettre de savoir si une image StreetView existe pour un lieu donné et de retourner le cas échéant cette image StreetView.
- **Geocoding API** va permettre de savoir si l'endroit cliqué sur la carte par un utilisateur retourne une adresse réelle et si c'est le cas, afficher cette adresse.

- **Google Places API** va permettre de retourner un certain nombre de restaurants pour une zone donnée et obtenir des informations spécifiques sur ces restaurants (note, nombre d'avis, commentaires etc.).
- **W3C Geolocation API** va retourner la position de l'utilisateur. Cette fonctionnalité est uniquement disponible dans des environnements sécurisés. Il est donc nécessaire de sécuriser le site à l'aide du protocole HTTPS.

CDN

- **Font Awesome** va fournir des icônes.
- **Google Fonts** va fournir les polices de caractères.

B. Architecture

L'architecture MVC

L'application s'appuie sur une architecture MVC (Modèle - Vue - Contrôleur). L'objectif de cette architecture est de séparer la logique du code en trois parties distinctes :

- **Le modèle.** Il contient la logique métier et l'état courant de l'interface pendant le cycle de dialogue avec l'utilisateur.
- **La vue.** Elle porte la logique de présentation. Elle affiche les données du domaine et reçoit les interactions de l'utilisateur dont elle délègue la gestion au contrôleur.
- **Le contrôleur.** Il reçoit les interactions de l'utilisateur de la vue et les traite en modifiant le modèle. Il assure le lien entre le modèle et la vue.

Le patron d'architecture MVC fonctionne en cycle. L'utilisateur interagit avec les composants de la vue qui sont à sa disposition. Cela déclenche la création d'événements qui sont envoyés au contrôleur. Le contrôleur vérifie la conformité des interactions et en déduit les modifications à apporter au modèle qui les gère en fonction de ses règles métiers. Les modifications du modèle sont ensuite signalées à la vue qui se met à jour en conséquence. Cette architecture présente plusieurs avantages : clarté, lisibilité et modularité du code, meilleure maintenabilité de l'application, facilité de mise à jour et une séparation claire des responsabilités des méthodes pour une meilleure cohésion et un couplage plus faible.

Adaptation du pattern aux besoins du projet

Ici, cette architecture est quelque peu modifiée pour permettre l'intégration dans l'application de l'objet Map, disponible grâce à l'API Maps Javascript. Cet objet dispose de méthodes relatives à des données métiers (coordonnées géographiques, objets géographiques etc) et des méthodes propres à sa représentation dans le DOM (carte, markers, infowindows). Il s'agit d'un **MVCObject**. Ainsi, pour cette application, l'architecture Modèle-Vue-Contrôleur évolue vers un motif Modèle-Map-Vue-Contrôleur, où le contrôleur continue à assurer le lien mais cette fois-ci entre le modèle, la vue et la map.

Description des classes de l'application

Dans le cadre de notre application, la classe **Model**, contenue dans le fichier Model.js, représente le modèle de l'application. Elle définit les méthodes de création et de mises à jour des restaurants en mémoire, et ce, grâce à la classe **Store** contenue dans le fichier Store.js qui joue le rôle de base de

données. Ici, le stockage ne se fait pas dans une véritable base de données mais dans le stockage local du navigateur.

La classe **View**, contenue dans le fichier View.js, représente la vue de l'application. Elle écoute différents événements sur la page, sauf ceux qui interviennent sur la carte. Elle transmet les nouvelles valeurs du filtre, grâce à la classe **Slider** disponible dans le fichier slider.js. Elle transmet également l'ajout d'un nouvel avis ou d'un nouveau restaurant. Elle met à jour l'apparence de la page (hormis la carte), grâce à la classe **Template** contenue dans le fichier Template.js.

La classe **Map**, disponible dans le fichier map.js, représente la carte et ses composants. Elle définit l'écoute des événements sur la carte, la mise à jour des marqueurs et des infowindows mais elle permet aussi d'obtenir différentes données métiers tels que les coordonnées géographiques, les restaurants et le détail des avis des restaurants.

La classe **Controller**, présente dans le fichier Controller.js, fait le lien entre le modèle, la map et la vue. Elle va interpréter les événements qui se produisent sur la page pour pourvoir donner des instructions au modèle et en retour, demander à la vue et à la map de mettre à jour la page.

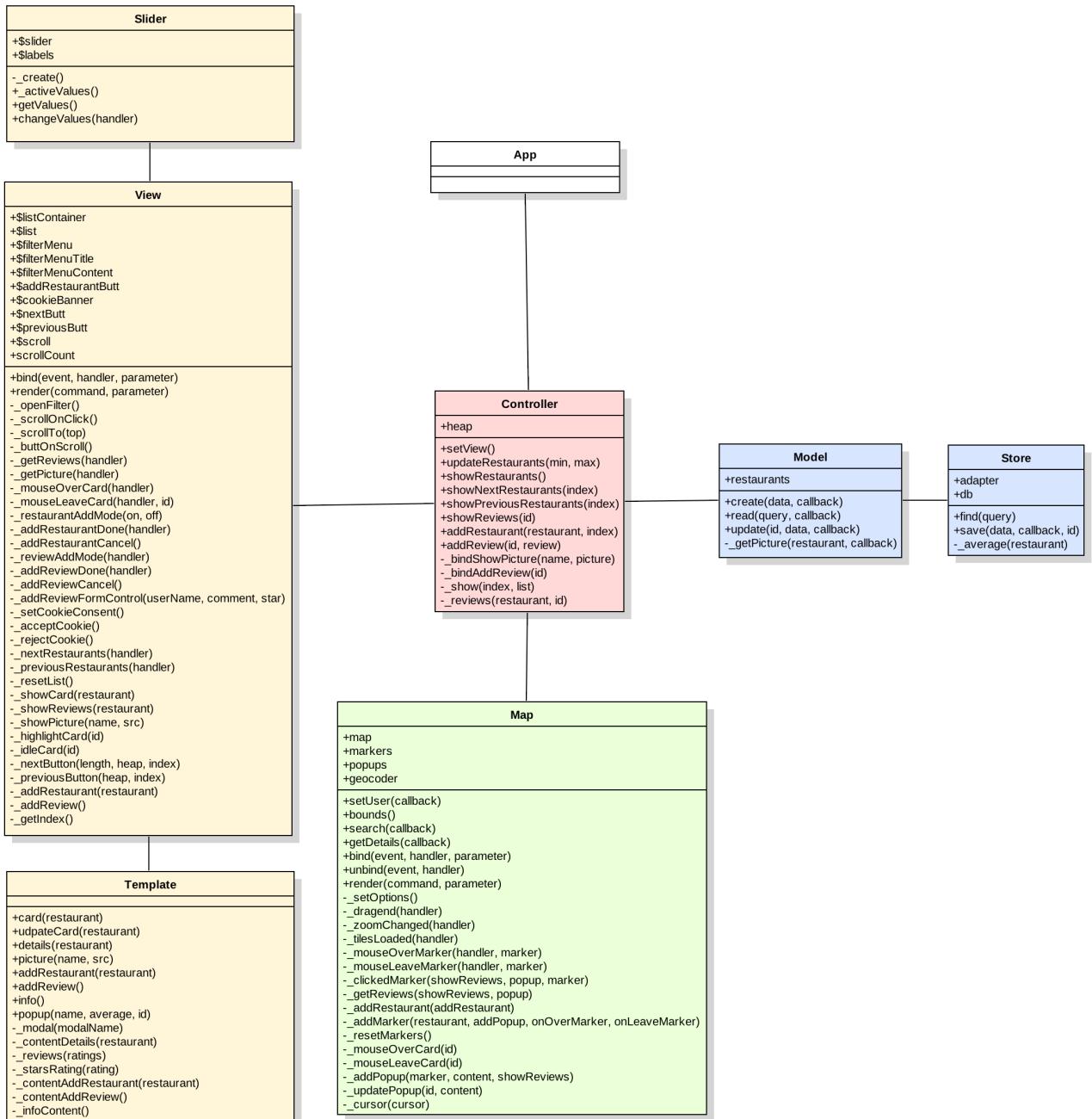
Lors de sa propre instanciation, la classe **App**, contenue dans le fichier app.js, génère une instance de chacune de ces classes.

Enfin, un fichier helpers.js déclare des méthodes globales pour la gestion des cookies.

Les méthodes de chacune des classes de l'application sont détaillées ici :

https://gaelleperchoc.github.io/OC_P7/.

C. Diagramme de classes

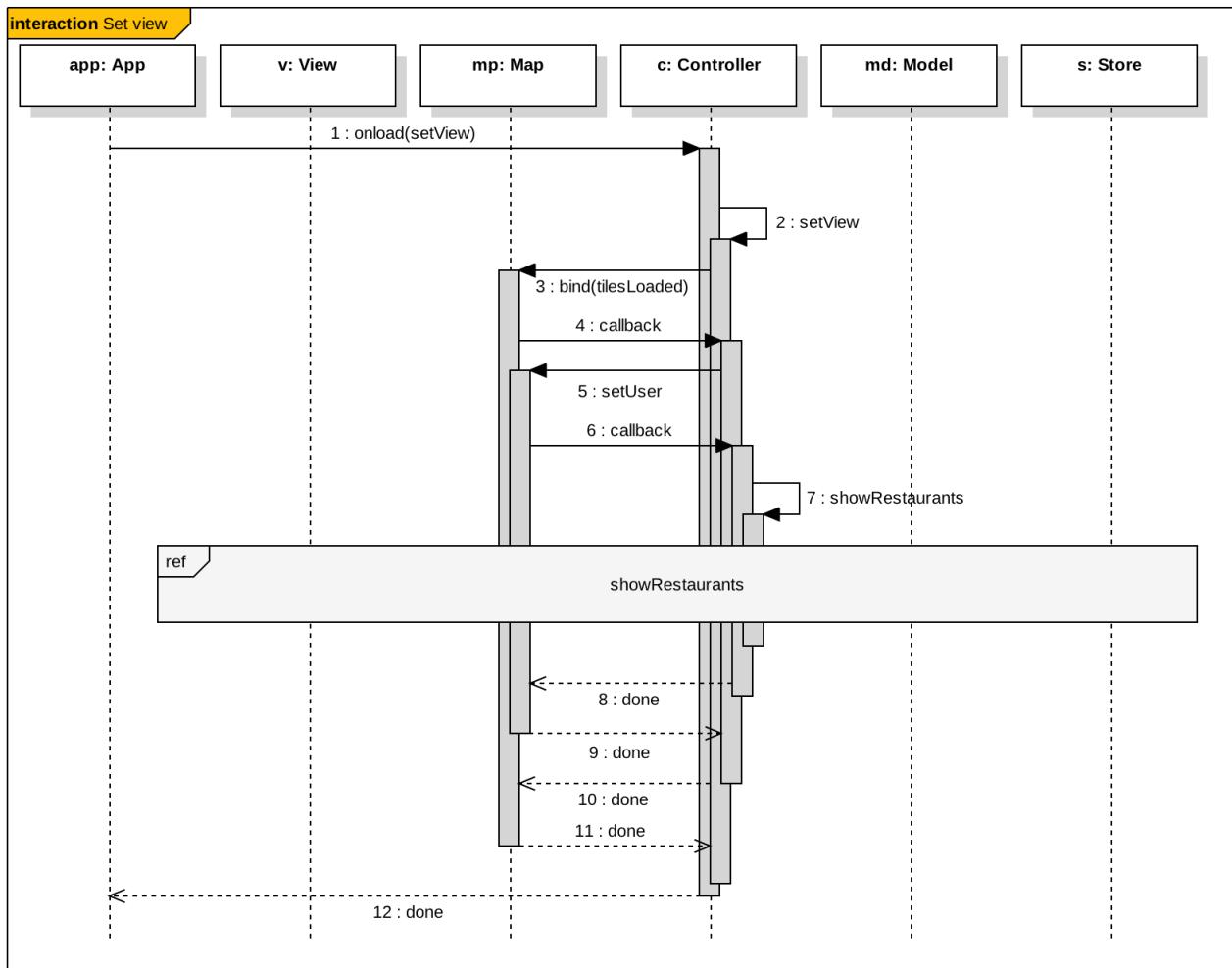


D. Diagrammes de séquence

Dans cette partie, un ensemble de diagrammes de séquence illustre pour chacune des fonctionnalités de l'application la manière dont les différents composants du système interagissent entre eux et avec l'utilisateur (cf. partie D.1 Fonctionnalités). Dans un souci de lisibilité, certaines méthodes font l'objet de diagrammes de séquences qui leur sont propres (cf. partie D.2 Fragments).

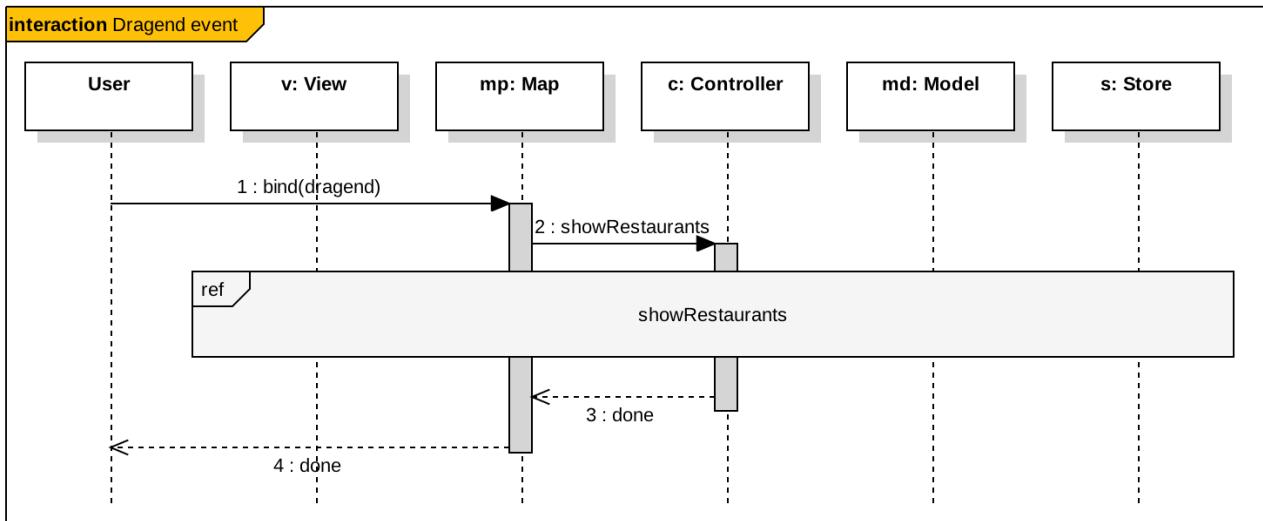
D.1. Fonctionnalités

Mise en place de la vue¹

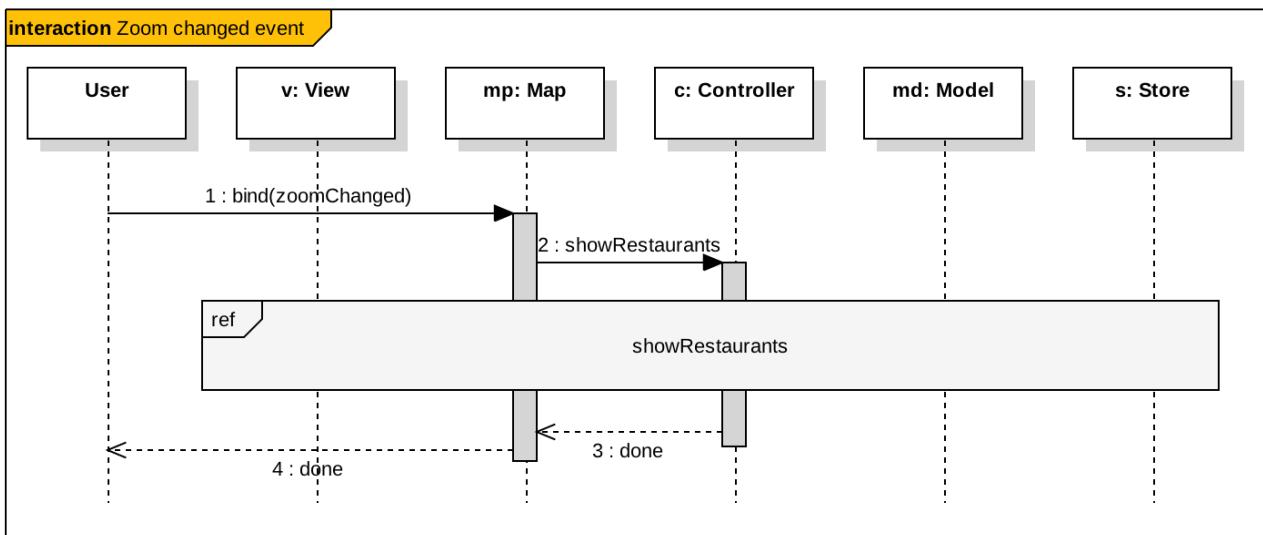


¹ Le fragment de séquence « showRestaurants » est détaillé p.35.

Dragend event²



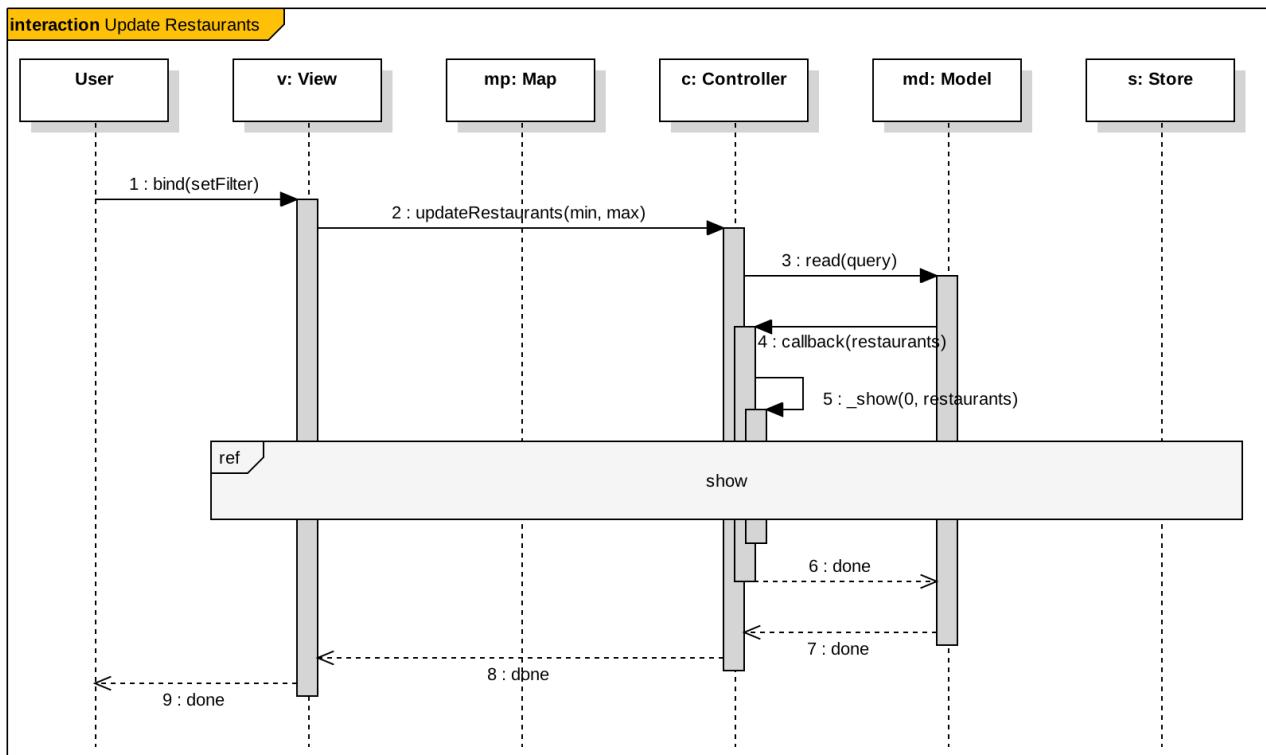
ZoomChanged event³



² Le fragment de séquence « showRestaurants » est détaillé p.35.

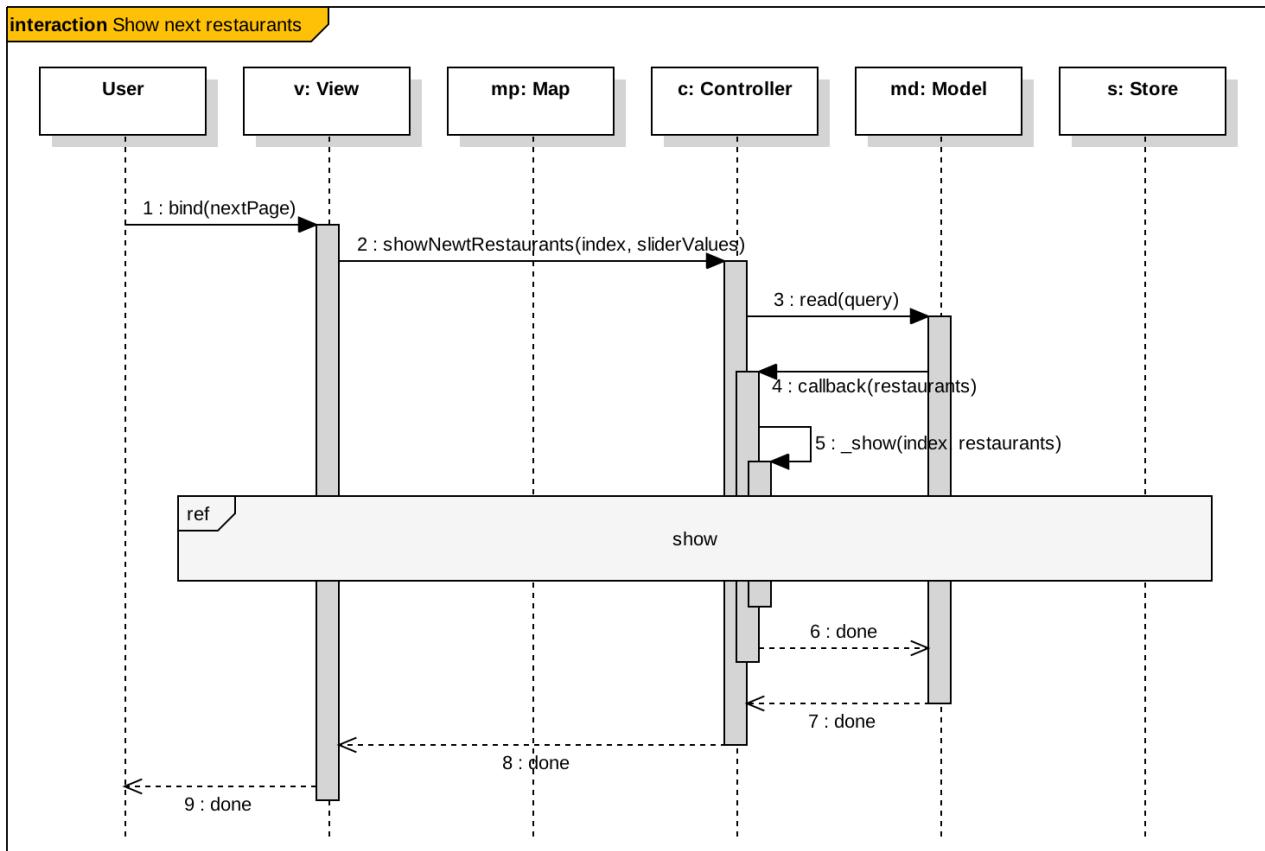
³ Idem.

Filtrer les restaurants par note⁴

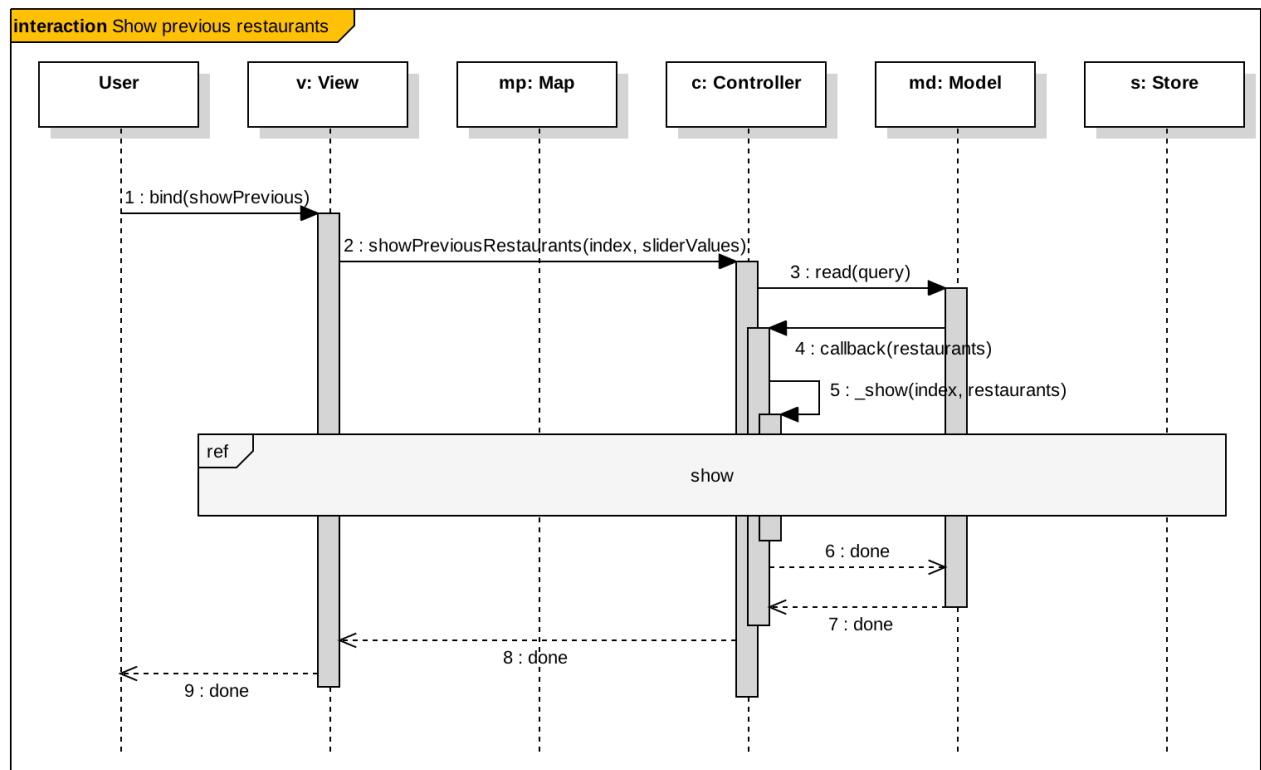


⁴ Le fragment de séquence « show » est détaillé p.34.

Montrer les restaurants de la page suivante⁵



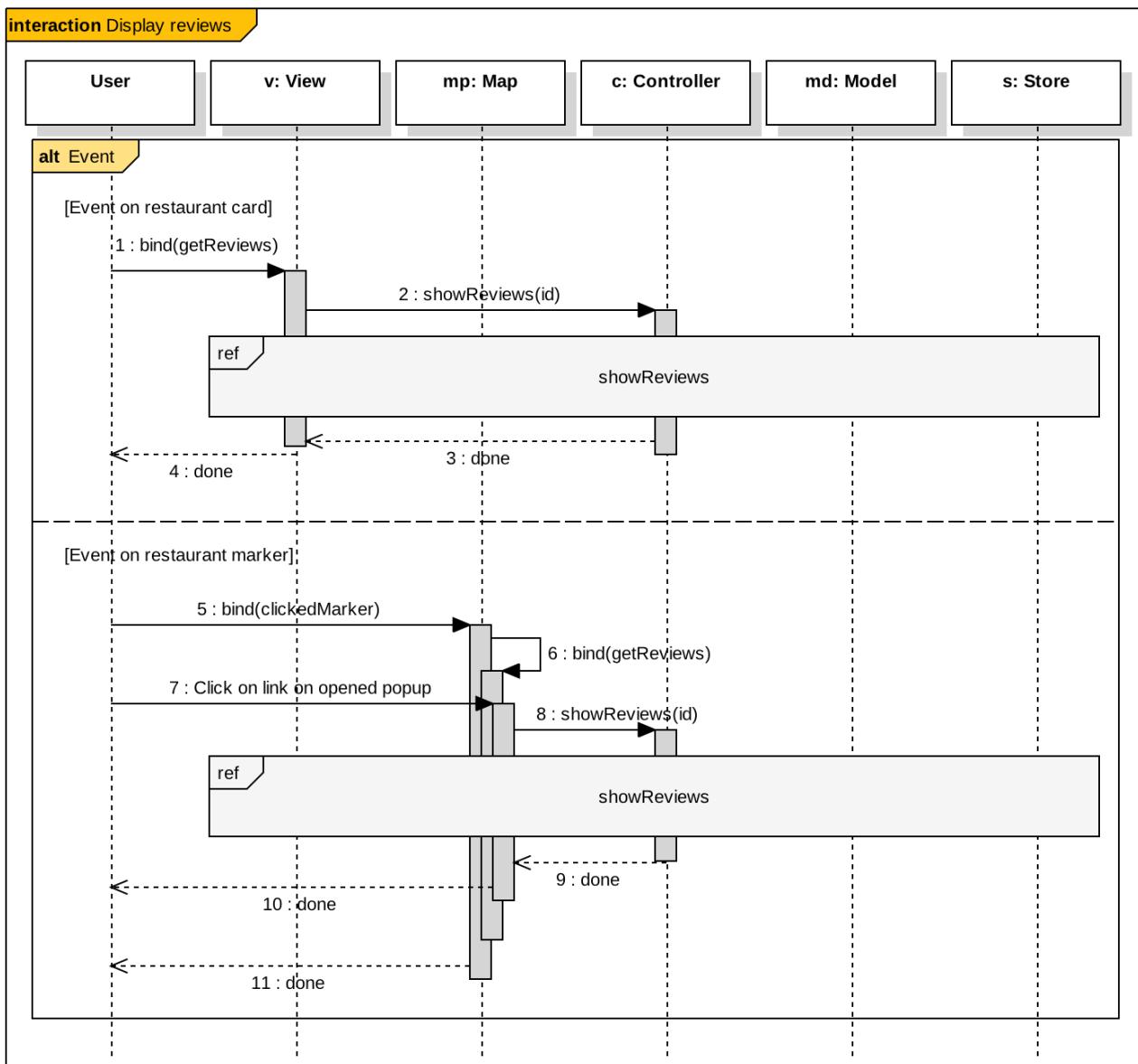
Montrer les restaurants de la page précédente⁶



⁵ Le fragment de séquence « show » est détaillé p.34

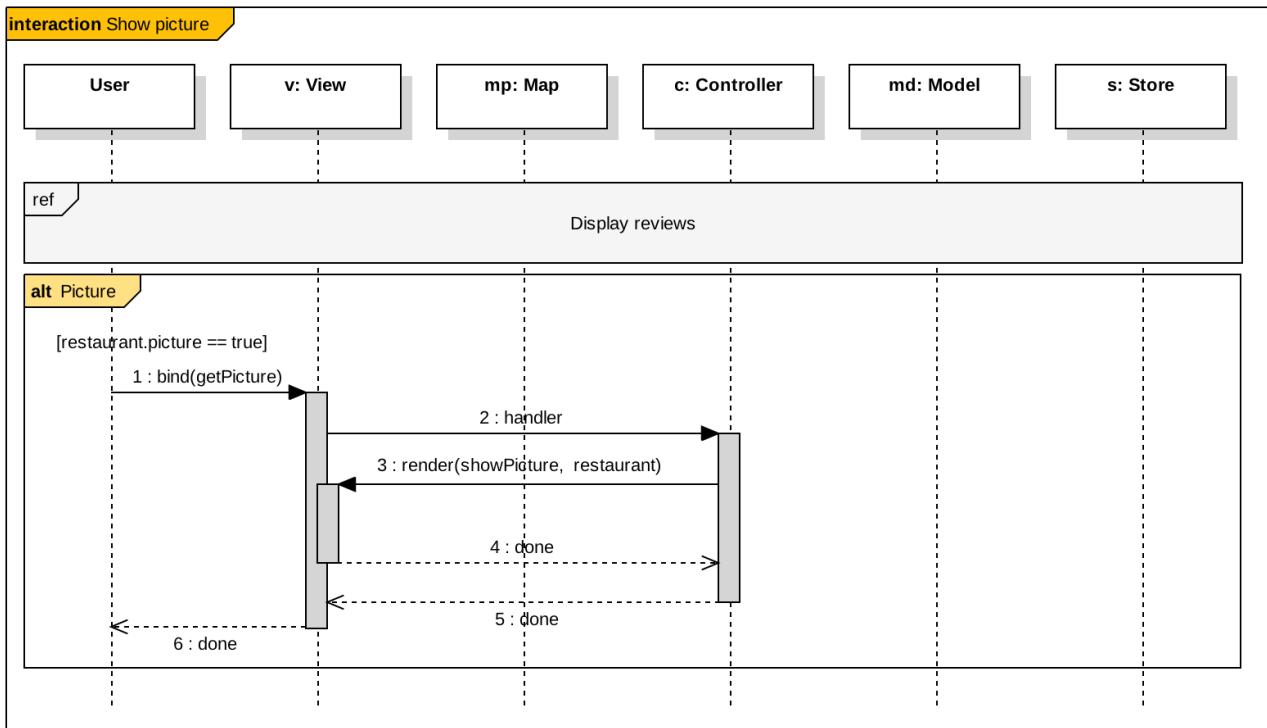
⁶ Idem.

Afficher les avis du restaurant⁷



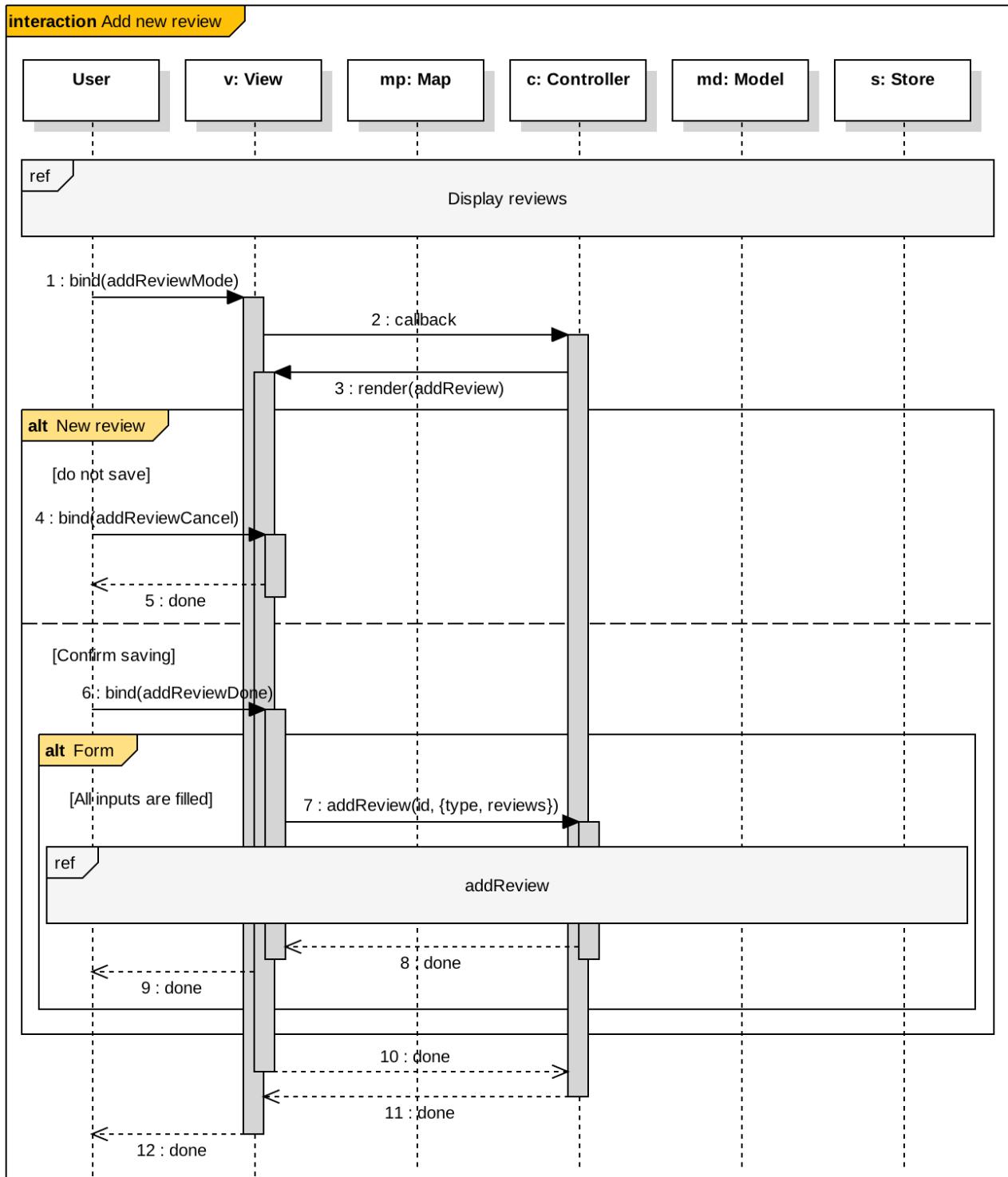
⁷ Le fragment de séquence « showReviews » est détaillé p.36.

Afficher la photo du restaurant⁸



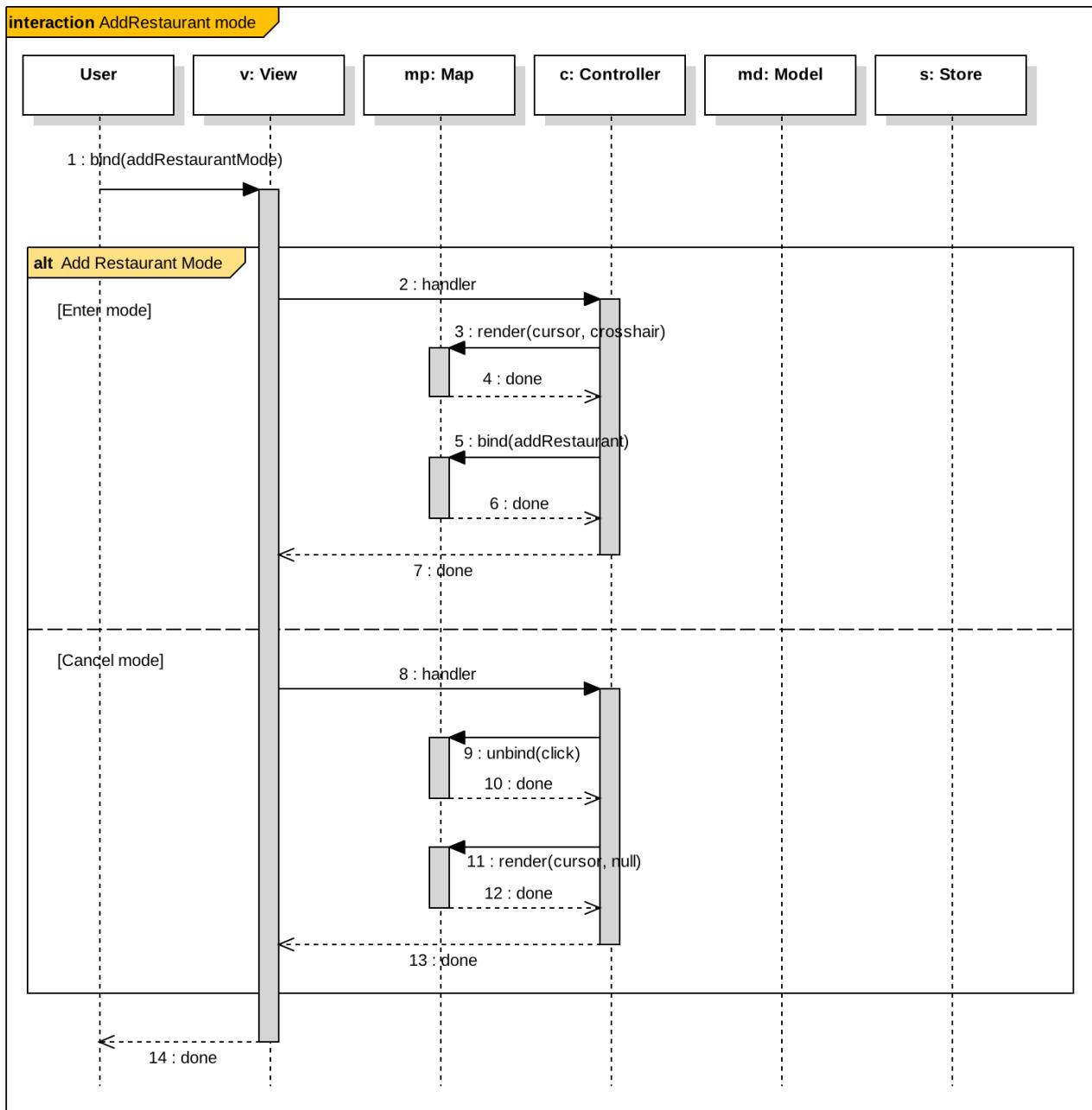
⁸ La séquence « Display Reviews » est présentée à la page précédente.

Ajouter un nouvel avis⁹

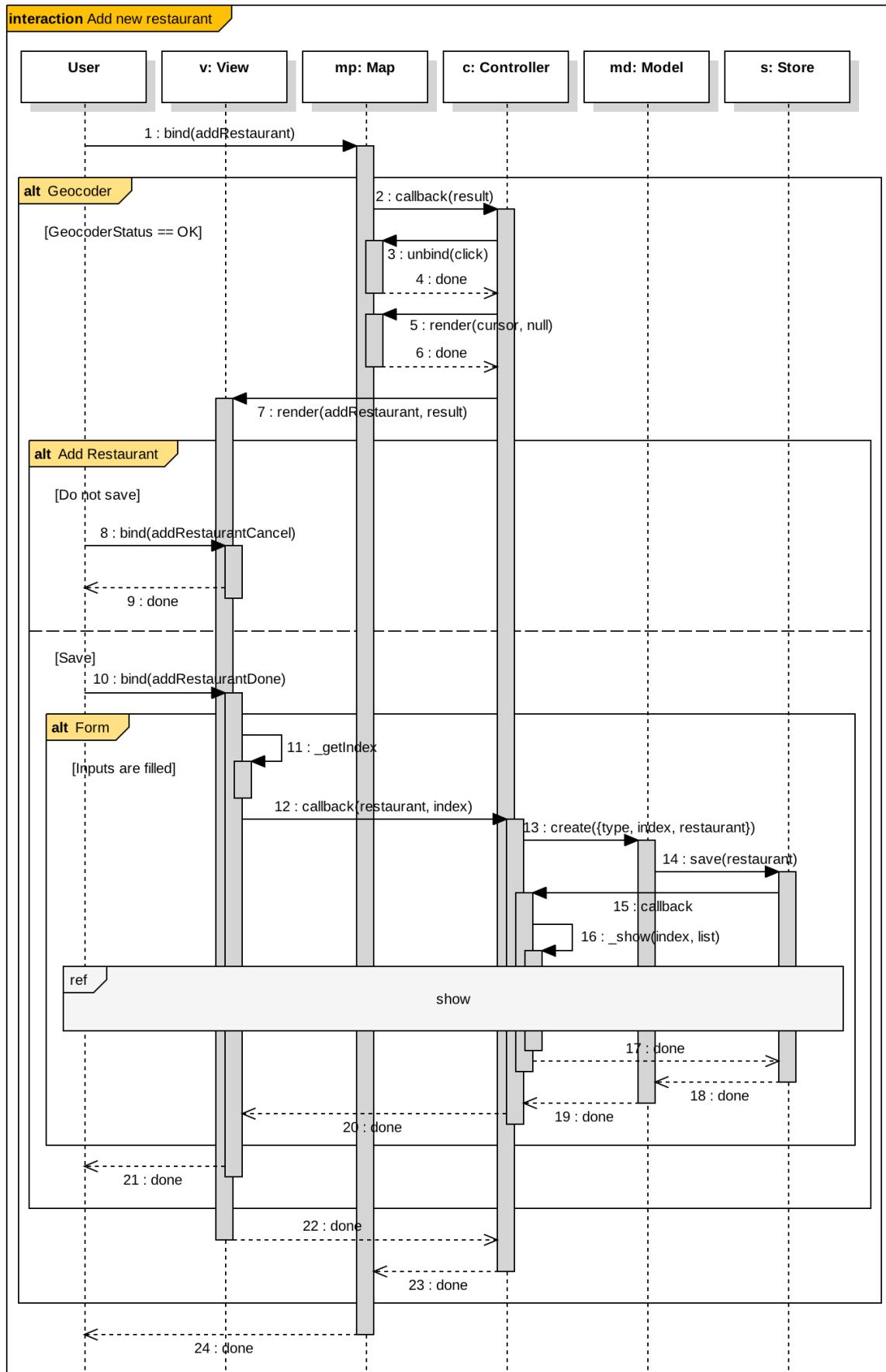


⁹ La séquence « Display reviews » est présentée p. 29.
Le fragment de séquence « addReview » est présenté p.38.

Entrer dans le mode pour ajouter un nouveau restaurant



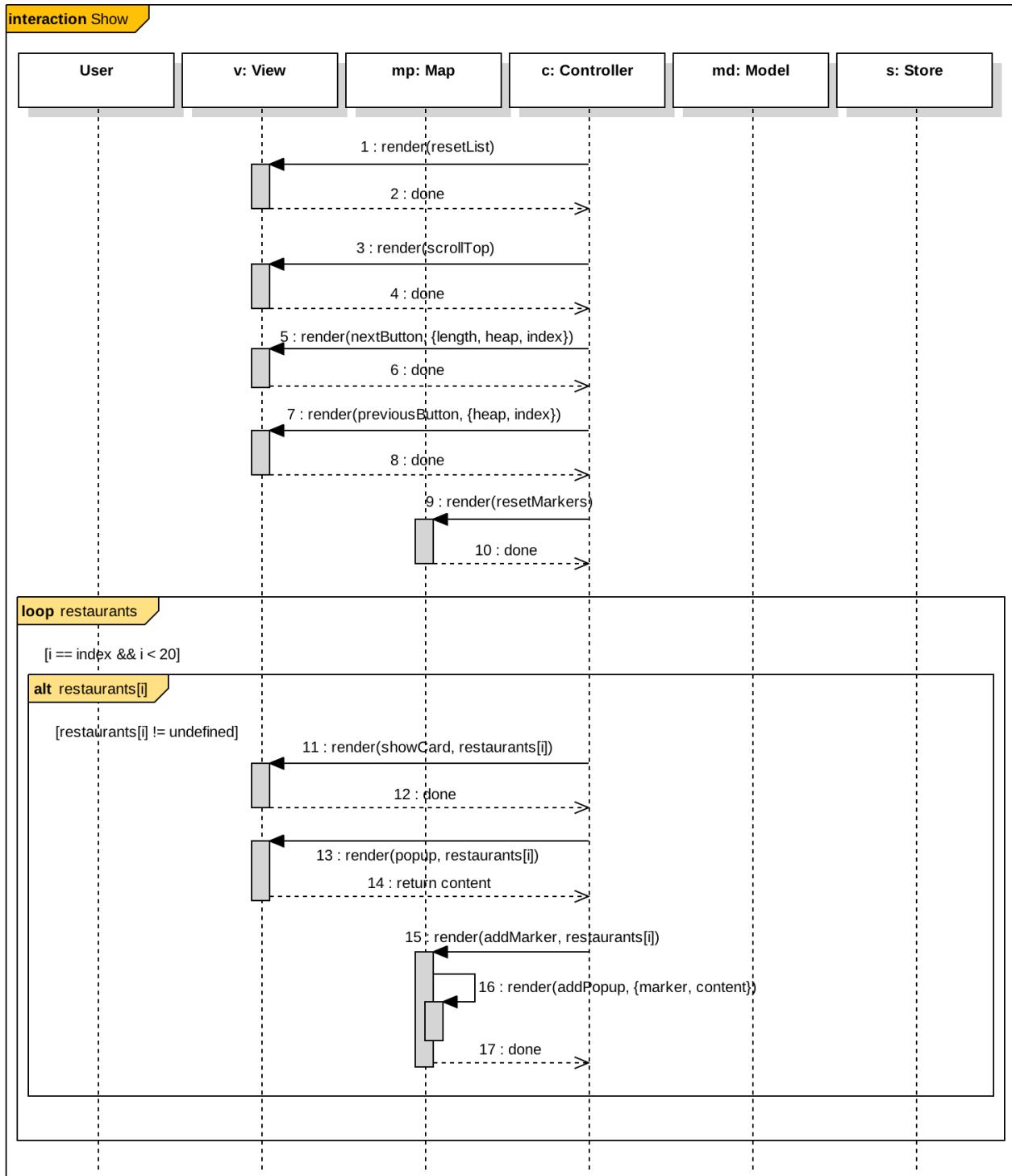
Ajouter un nouveau restaurant¹⁰



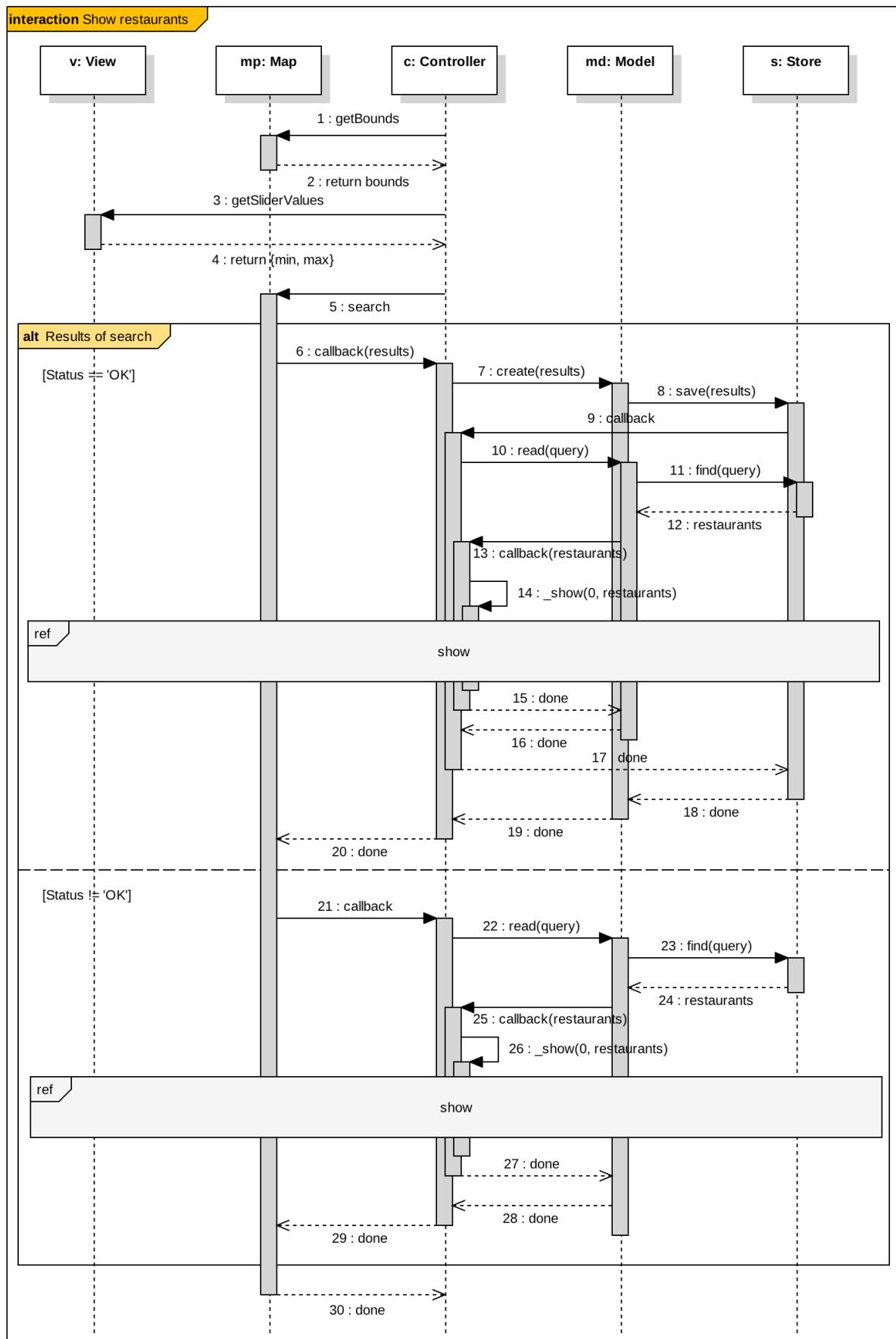
¹⁰ Le fragment de séquence « show » est présenté p.34.

D.2. Fragments

Mettre à jour la liste et les marqueurs des restaurants

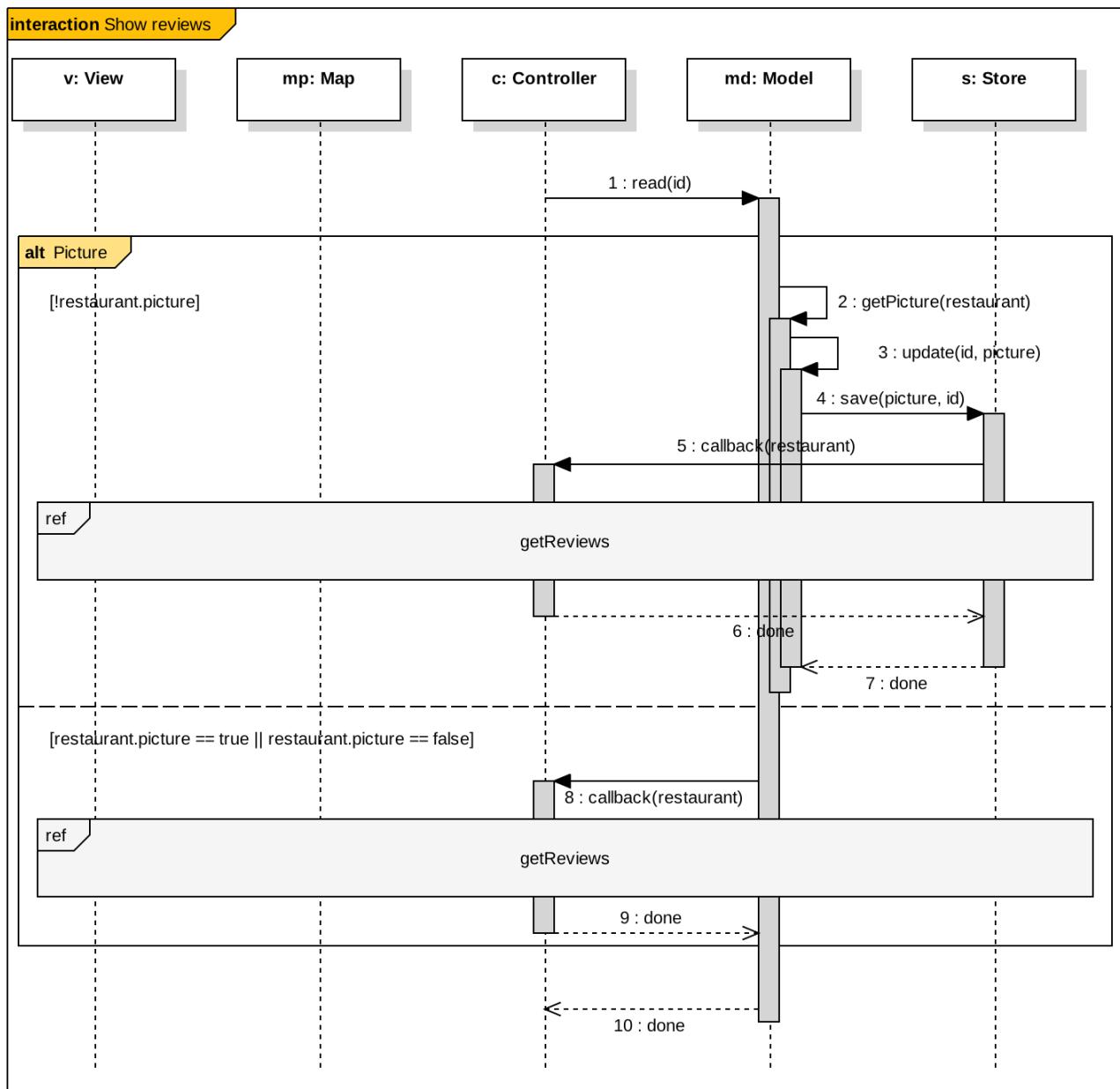


Afficher la liste et les marqueurs des restaurants¹¹



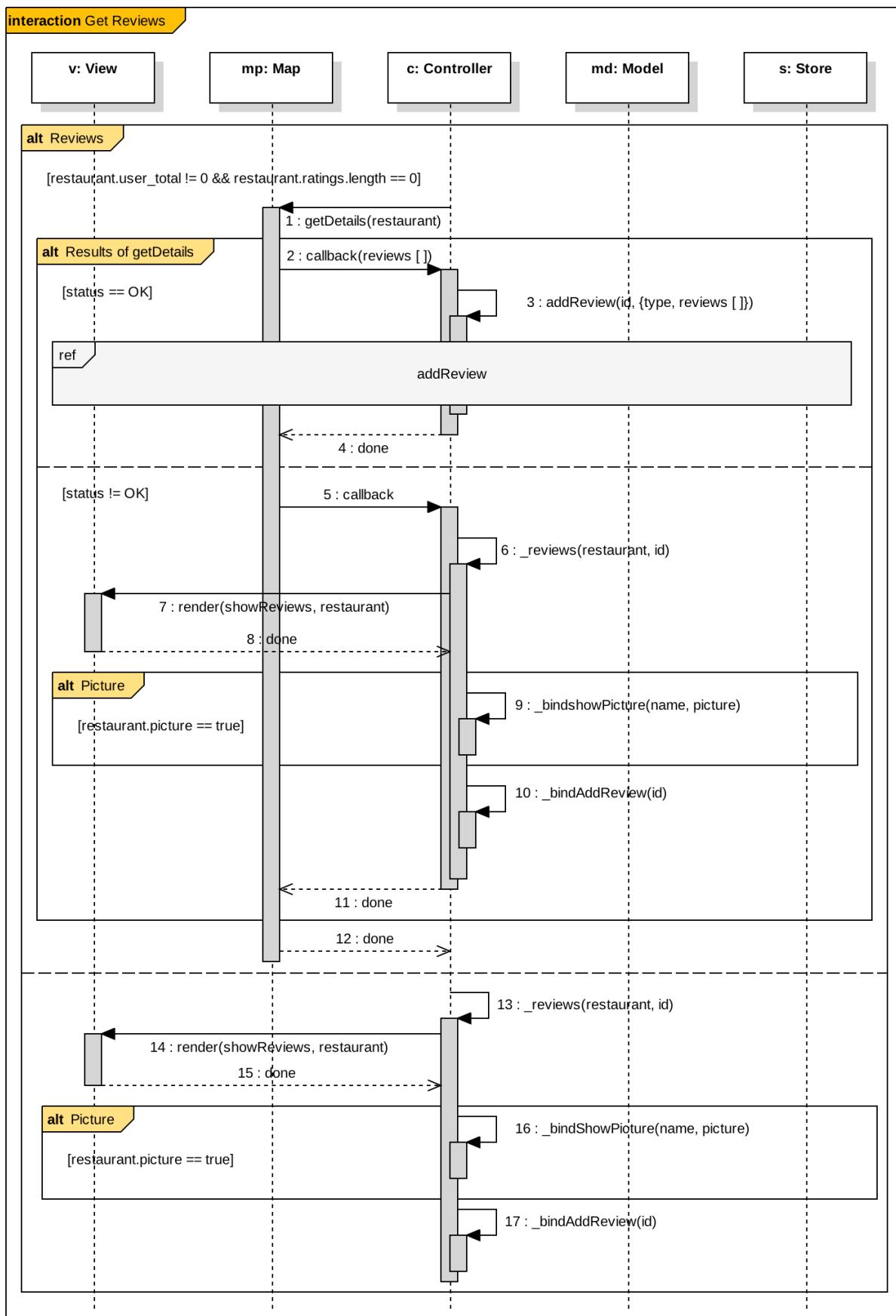
¹¹ Le fragment de séquence « show » est présenté p.33.

Mise à jour des propriétés du restaurant avant d'afficher les avis¹²



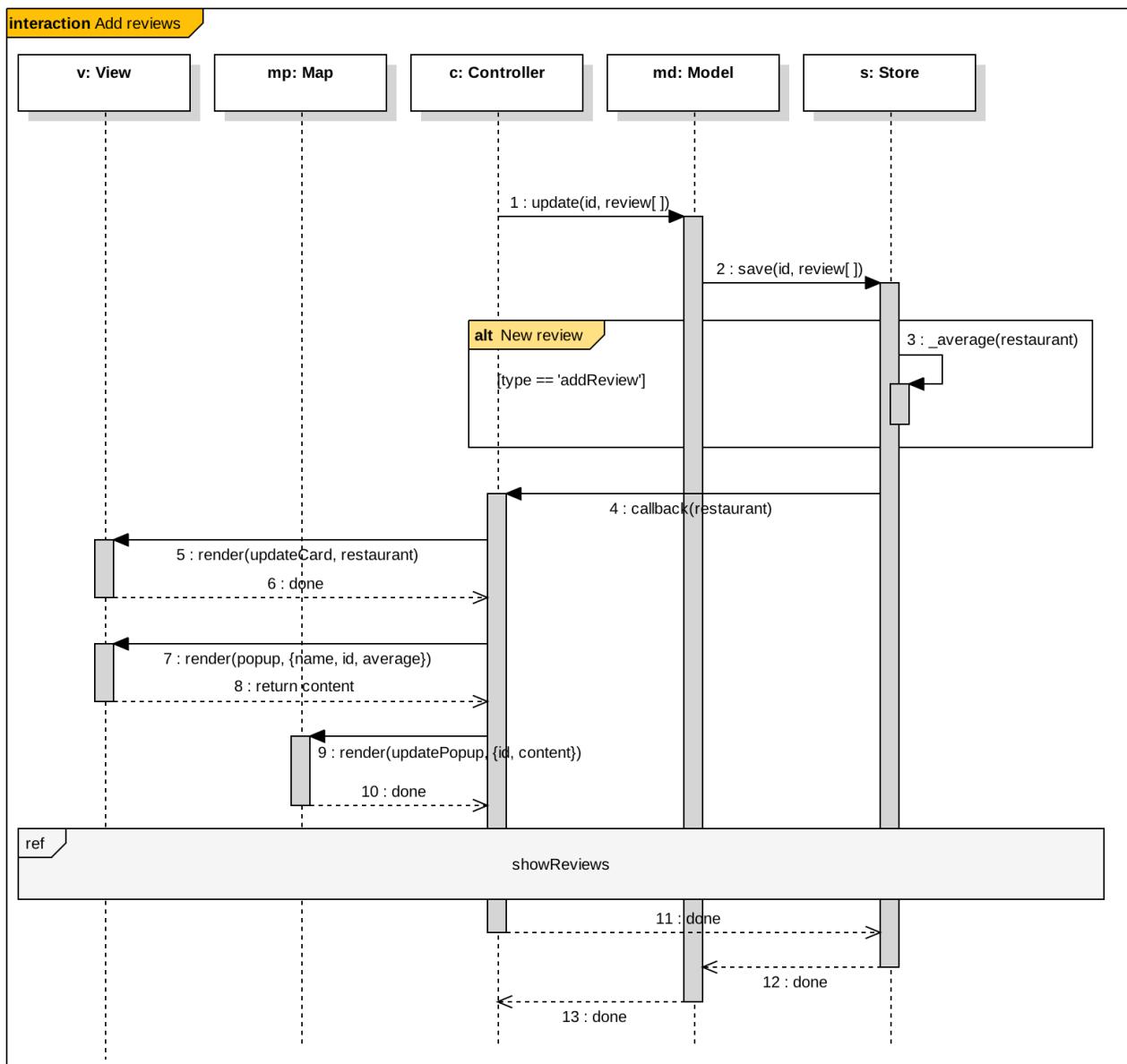
¹² Le fragment de séquence « getReviews » est présenté p.36.

Obtenir les avis du restaurant¹³



¹³ Le fragment de séquence « addReview » est présenté p.37.

Ajouter un ou plusieurs avis¹⁴



¹⁴ Le diagramme de séquence « showReviews » est présenté p.36.

IV. Réflexion autour du projet : limites et opportunités d'évolution

Quotas et budgets

Depuis juin 2018, les services de Google Maps sont devenus payants dans les applications tierces. Un crédit mensuel de 200\$ est alloué à chaque utilisateur par la firme. Afin d'éviter tout dépassement inopiné et assurer un suivi de la consommation du crédit, il est nécessaire de mettre en place un budget et des alertes pour le projet, pour prévenir ainsi l'évolution de la consommation du crédit. Les quotas sont également un outil primordial pour une gestion mesurée du crédit. Ils permettent de définir un nombre de requêtes par jour, globalement et par utilisateur, pour chacune des API utilisées dans le projet. Les budgets et les quotas sont à réfléchir et définir en amont du projet.

Limitation et pertinence des résultats de la recherche par zone

La recherche par zone de Google Places offre un nombre limité de résultats. Une requête retourne jusqu'à 20 établissements. Il est toutefois possible d'obtenir davantage de résultats, car chaque recherche par zone contient en réalité jusqu'à 60 établissements, répartis sur 3 pages. Pour obtenir l'ensemble des résultats, il est nécessaire de réaliser une recherche en exploitant un paramètre spécifique généré si plusieurs pages existent. La recherche générera alors jusqu'à 3 requêtes. Elle distribuera les résultats par paquet de 20, avec un délai de latence de 2 secondes entre chaque paquet. Dans le cadre de ce projet, nous n'avons pas réussi à exploiter l'ensemble des résultats de chaque recherche sans que ce soit dommageable pour la performance de l'application ou pour la gestion des quotas établis en amont. Nous nous en sommes donc tenus aux 20 premiers résultats afin d'offrir une meilleure expérience utilisateur. Toutefois, la recherche par zone offre peu d'options pour personnaliser et filtrer les résultats et la question de leur pertinence se pose quand ceux-ci renvoient à Paris, par exemple, vers des hôtels(-restaurants) quand il s'agit en premier lieu d'une recherche de restaurants.

Alternatives à Google Maps

Afin de dépasser ces questions de quotas, budgets et de pertinence, des alternatives à Google Maps existent, notamment avec le projet libre OpenStreetMap qui dispose d'une large communauté et de solides ressources. En combinant des outils comme Leaflet, Mapillary et Overpass Turbo, il serait possible d'approcher un résultat proche, si ce n'est équivalent, du projet actuel.