Zingat Frontend Challenge

Requirements

1) Acceptance Criteria

- Project must be hosted on a public git repo, such as Github, Gitlab, Bitbucket etc.
- There must be a documentation that contains information on how to build project on local.

2) General Backend Requirements

- Project must be Node.js application
- Project must use a Node.js MVC framework such as Express.js, Adonis.js or something like Koa.js that can be extended using appropriate packages for templating, routing etc.
- Project must use a templating engine such as Pug.js, Nunjucks, Swig etc.
- Project must have routing capability.

3) General Client Requirements

- Styles must be written using a CSS preprocessor such as Stylus, SASS or LESS.
- Styles must be responsive. Keep desktop design on resolutions >= 960px and mobile version < 960px.
- Any font, color, margin, padding etc. can be used when writing styles but markup hierarchy must follow designs on mockups.

4) Static Assets Requirements

- The design must have a valid/semantic HTML5 and use a CSS preprocessor like Stylus, LESS or SASS.
- Client-side javaScript code must be native (vanilla). We don’t want you to use any client-side library (jQuery, React, Vue etc.) or framework (Angular etc.)
- Use Gulp/Grunt tasks to compile Stylus/LESS/SASS, minify and combine them into one css file.
- Use Gulp/Grunt tasks to minify and combine javascript files into one JS file.
- Any plugins must be included to the project using NPM or YARN.

5) Login page

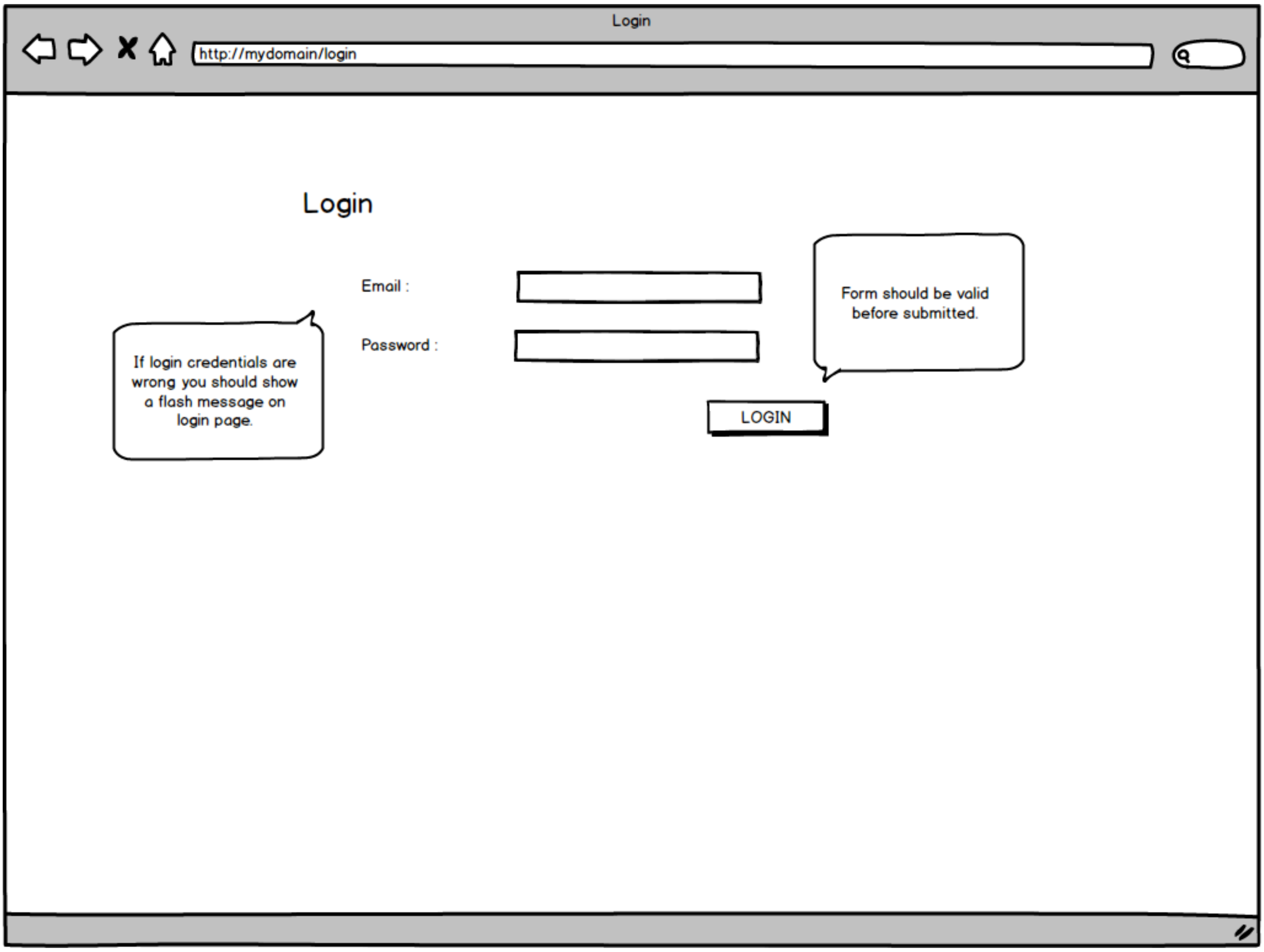
- Backend:
    - There must be a /login route.
    - There must be a Login controller.
    - Login POST must be handled, either using existing Login controller or another one for solely POST request.
    - We don't want you to implement a complex authentication & authorization. A manuel, dummy email and password can be controlled with provided values on appropriate place.
    - If login credentials are wrong you must handle this situation and return status.
- Client:
    - Layout must be designed according to mockups.
    - A client-side, basic form validation must be implemented. You can use any basic styles for error handling as you see fit.
    - If form is not valid, it must not be submitted.
    - If backend provides information about failed login action, Login page must be re-rendered and a flash message must be shown to user.

6) MyAlbums page

- Backend:
    - There must be a /myalbums route.
    - There must be a MyAlbums controller
    - MyAlbums controller must send request to https://jsonplaceholder.typicode.com/albums, and handle response data.
- Client:
    - Layout must be designed according to mockups.
    - After an album title is clicked, an AJAX request must be made on client-side to https://jsonplaceholder.typicode.com/albums/ALBUM_ID/photos and response must be used to render photo thumbnails.
    - If any thumbnail is clicked, a bigger photo must be shown at the bottom.

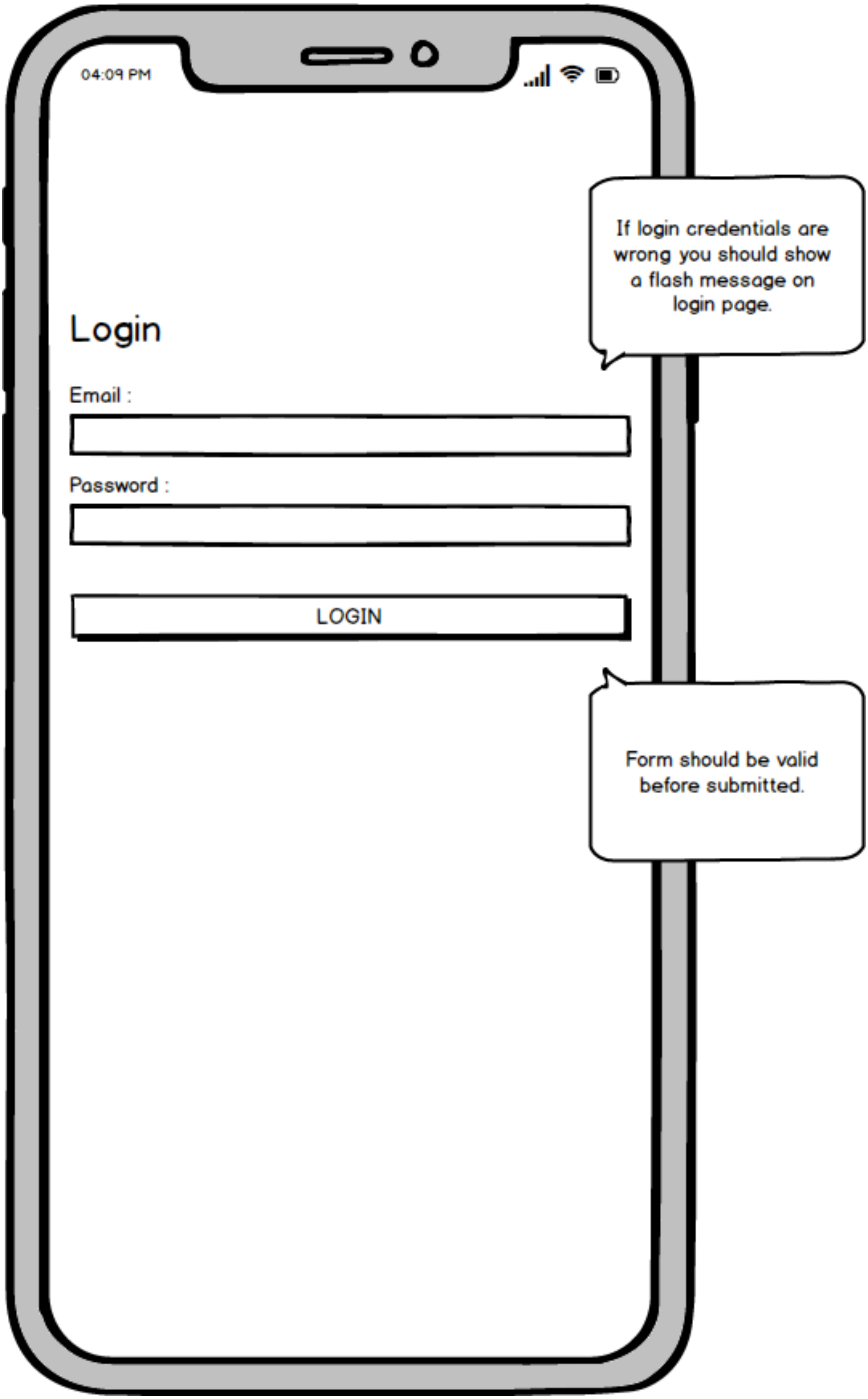
7) Nice to have :)

- A watcher mechanism for building static assets.
- Redis usage for data cache.
- Good documentation, both on public git repo and on code itself.
- Deployment of your project to a free cloud service, such as Heroku, Now etc.
- Continues Integration & Continues Deployment.
- Unit tests.



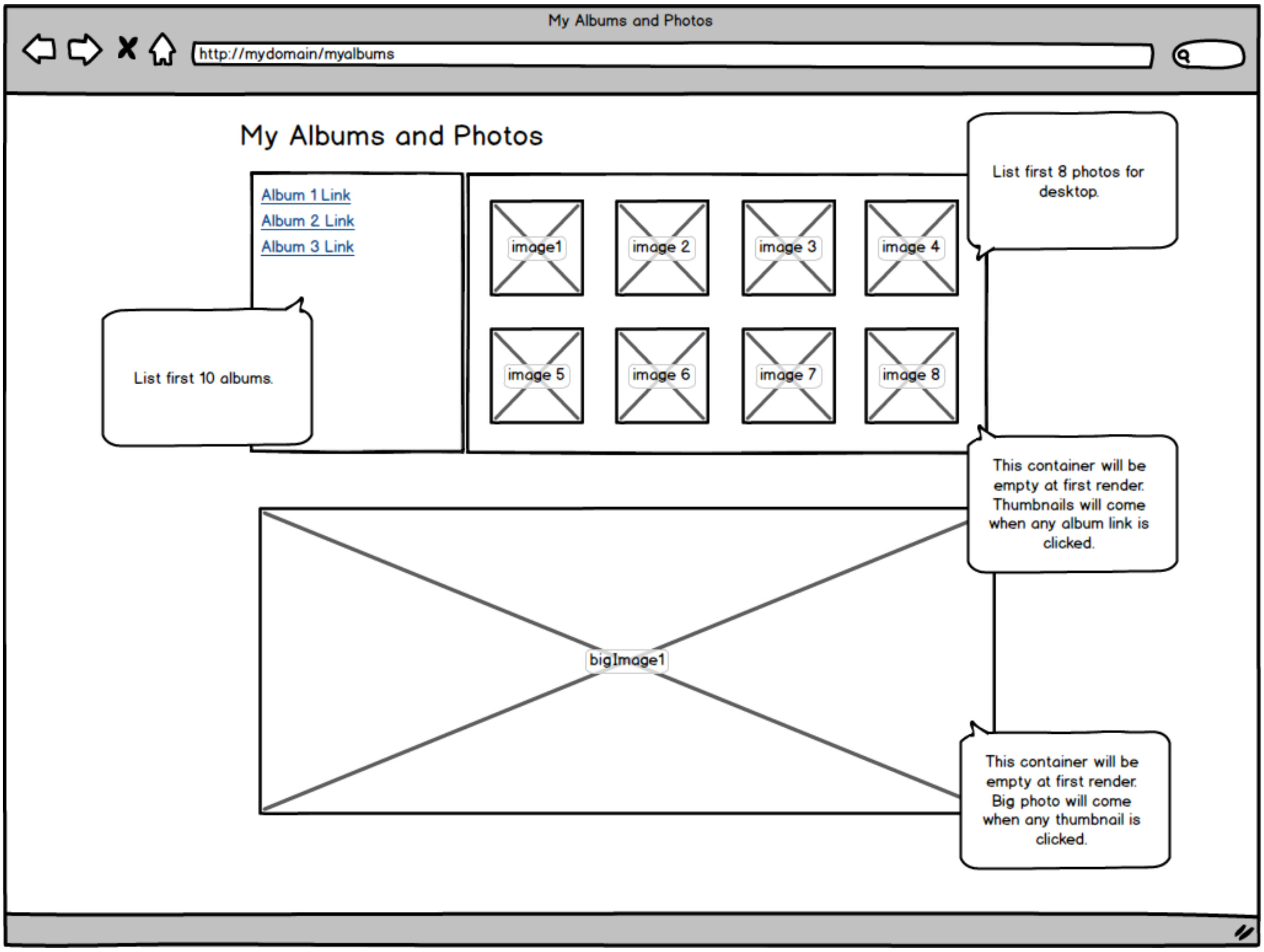
If login credentials are wrong you should show a flash message on login page.

Form should be valid before submitted.



If login credentials are wrong you should show a flash message on login page.

Form should be valid before submitted.

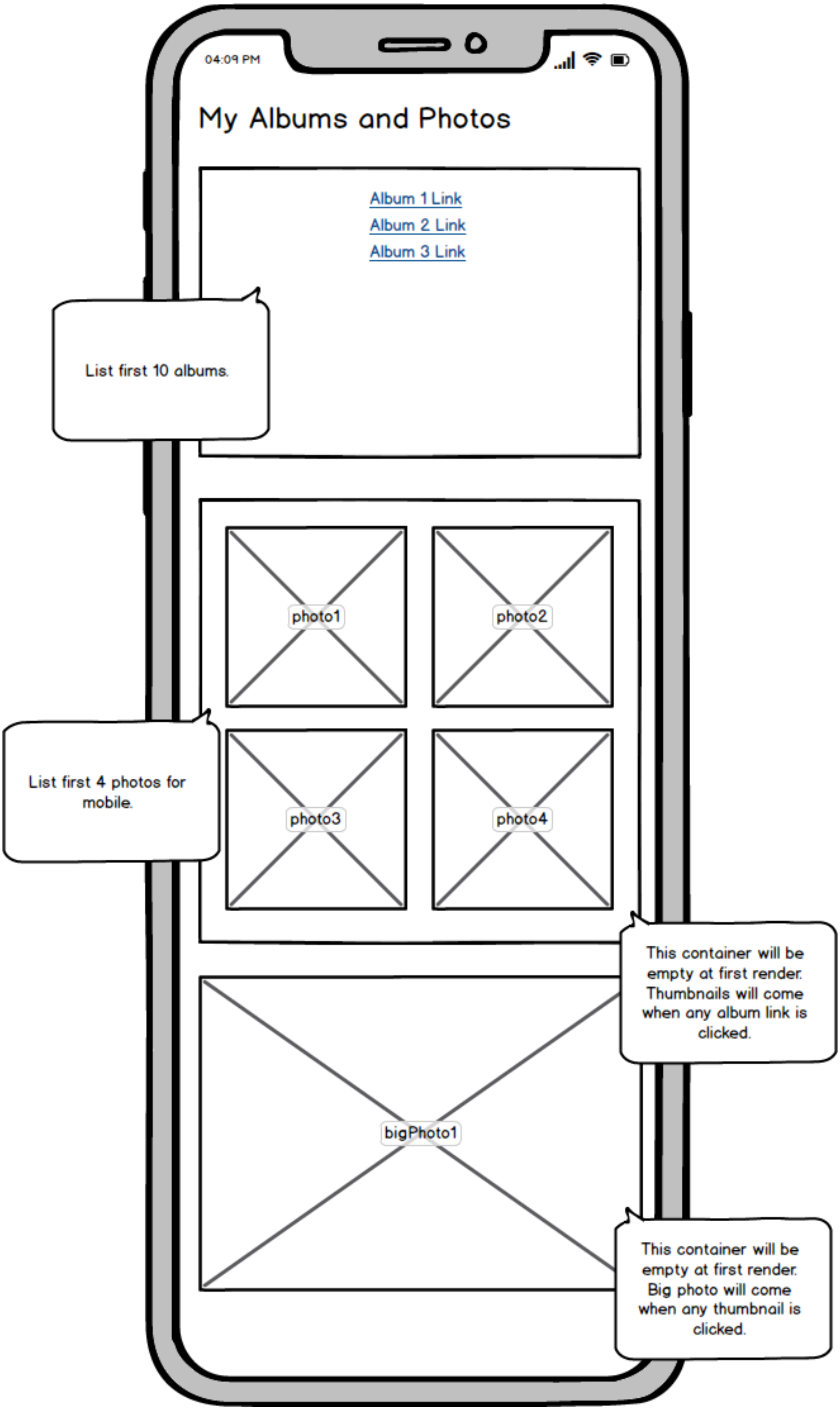


List first 8 photos for desktop.

List first 10 albums.

This container will be empty at first render. Thumbnails will come when any album link is clicked.

This container will be empty at first render. Big photo will come when any thumbnail is clicked.



List first 10 albums.

List first 4 photos for mobile.

This container will be empty at first render. Thumbnails will come when any album link is clicked.

This container will be empty at first render. Big photo will come when any thumbnail is clicked.