پروژه ارزیابی توسعه‌دهنده فرانت‌اند
شماره ۰۰۲ نسخه ۱

1. create a basic react app[novice]

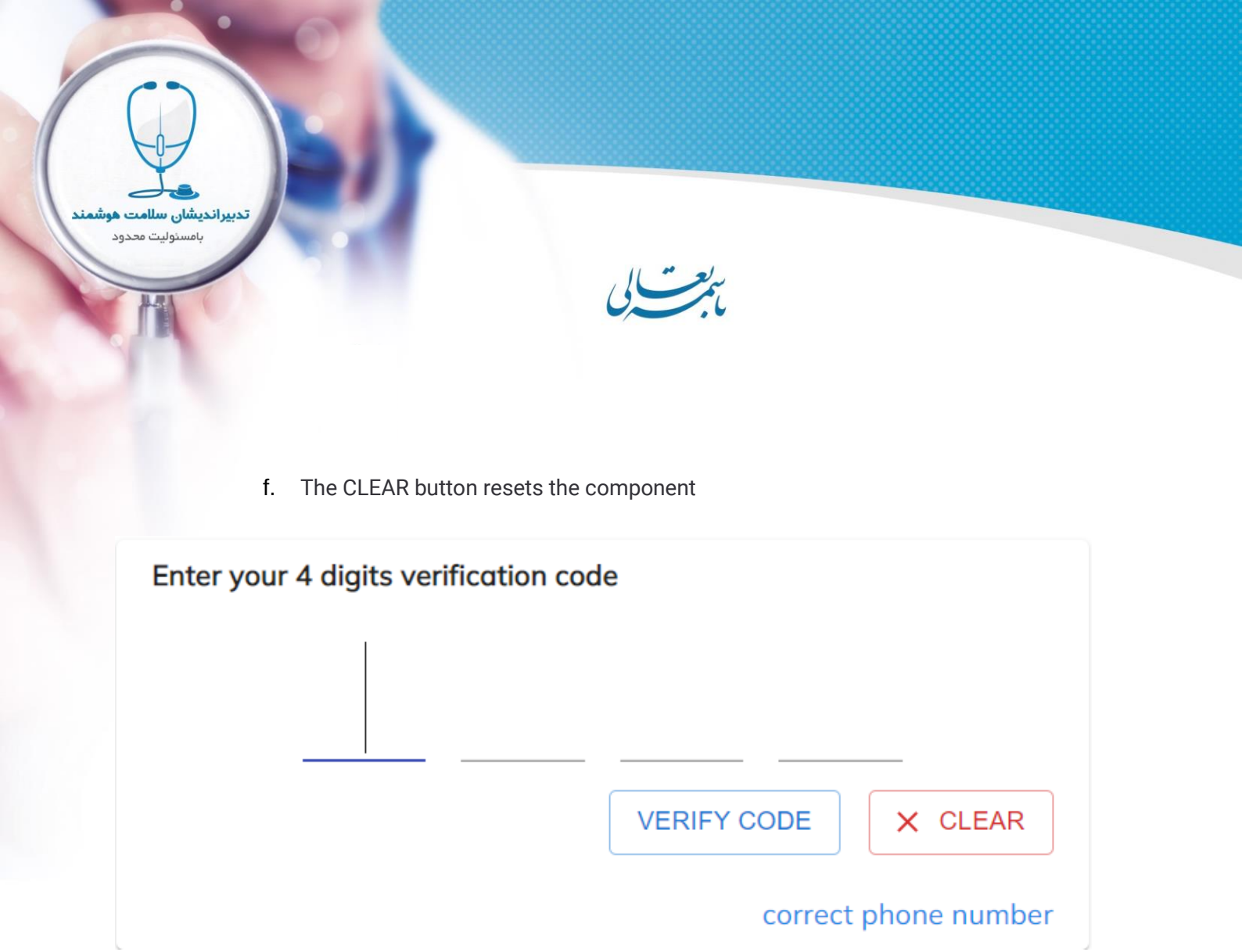
a. just initialize a React-JS application

2. create 3 routes[novice]

a. /products
b. /users
c. /verification
d. combination of CLASS and FUNCTIONAL components

3. create a verification component[practitioner]

a. In the verification component, we will have 4 number fields (You made it in the previous step).
b.
c. it does accept numbers
d. The focus should initially be on the first number field
e. As each number field is filled, the focus will move to the next number field



f. The CLEAR button resets the component

4. Linter command[practitioner]

a. in package.json create a simple linter command, we can run it with this command : **yarn run lintmyprogram**
b. note: you can use Danger JS, …

5. display a dialog[practitioner]

a. display a dialog with ctrl+shift+f2
b. there is a textfield in this dialog
c. we want to focus on this text field with ctrl+shift+f3

باسمه تعالی

6. using Redux[practitioner]

a. in the previous step you created a textfield. Now we need to send text field data to redux and read from Redux, then display the content of text field under the textfield.

7. code splitting[expert]

a. split the code based on routes

8. PWA[expert]

a. convert your app to a PWA app. We need these two features:
b. the app MUST be installable
c. the program must be able to store its static resources (chunks you created in the previous step) in cache

9. E2E test[expert]

a. write just one E2E test with cypress. you should write a test for navigations you created in first step

10. micro front-End[ninja]

a. create a simple Button component in another repository and use it as a micro front-End in to your project

باسمه تعالی

11. Dockerfile[ninja]

a. create a dockerfile for project

12. Pipeline[ninja]

a. create a simple pipeline for the project. and make the conditions on **main** branch.(CI/CD)

What you need to know

We would love for you to contribute and help us to find you today:) As a contributor, here are the guidelines we would like you to follow:

- How to send Project
- Which items should I complete?
- Question or Problem?
- Commit Message Guidelines
- language

How to send project

Please create a repository in your git account and invite this user :

1. Gitlab.com :@miladkhan
2. Github.com: @milad-kh

Which items should I complete?

There are four levels for issues [novice, practitioner, expert, ninja], once you complete issues of each level, we can determine your proficiency.

You can complete all or part of issues. The more issues are completed and confirmed, the more score you will receive.

Got a Question or Problem?

Do not open any issue for general support questions as we want to keep issues for extending the assessment process. Instead, we recommend using Stack Overflow to ask support-related questions. make sure to add the react tag.

Stack Overflow is a much better place to ask questions since:

- there are thousands of people willing to help on Stack Overflow
- questions and answers stay available for public viewing so your question/answer might help someone else
- Stack Overflow's voting system assures that the best answers are prominently visible.

Commit Message Format

This specification is inspired by and supersedes the [AngularJS commit message format][commit-message-format].

We have very precise rules over how our Git commit messages must be formatted. This format leads to **easier to read commit history**.

Each commit message consists of a **header**, a **body**, and a **footer**.

<header>
<BLANK LINE>
<body>
<BLANK LINE>

<footer>

The header is mandatory and must conform to the Commit Message Header format.

The body is mandatory for all commits except for those of type "docs". When the body is present it must be at least 20 characters long and must conform to the Commit Message Body format.

The footer is optional. The Commit Message Footer format describes what the footer is used for and the structure it must have.

Commit Message Header

```
<type>(<scope>): <short summary>
  │       │             │
  │       │             └⫸ Summary in present tense. Not capitalized. No period at the end.
  │       │
  │       └⫸ Commit Scope: animations|bazel|benchpress|common|compiler|compiler-
cli|core|
  │                       elements|forms|http|language-service|localize|platform-browser|
  │                       platform-browser-dynamic|platform-server|router|service-worker|
  │                       upgrade|zone.js|packaging|changelog|docs-infra|migrations|ngcc|ve|
  │                       devtools
  │
  └⫸ Commit Type: build|ci|docs|feat|fix|perf|refactor|test
```

The <type> and <summary> fields are mandatory, the (<scope>) field is optional.

Type

Must be one of the following:

- **build**: Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
- **ci**: Changes to our CI configuration files and scripts (examples: CircleCi, SauceLabs)

- **docs**: Documentation only changes
- **feat**: A new feature
- **fix**: A bug fix
- **perf**: A code change that improves performance
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **test**: Adding missing tests or correcting existing tests

Scope

The scope should be the name of the npm package affected (as perceived by the person reading the changelog generated from commit messages).

The following is the list of supported scopes:

- animations
- bazel
- benchpress
- common
- compiler
- compiler-cli
- core
- elements
- forms
- http
- language-service
- localize
- platform-browser
- platform-browser-dynamic
- platform-server
- router
- service-worker
- upgrade

There are currently a few exceptions to the "use package name" rule:

- packaging: used for changes that change the npm package layout in all of our packages, e.g. public path changes, package.json changes done to all packages, d.ts file/format changes, changes to bundles, etc.
- changelog: used for updating the release notes in CHANGELOG.md
- dev-infra: used for dev-infra related changes within the directories /scripts and /tools
- docs-infra: used for docs-app (angular.io) related changes within the /aio directory of the repo
- migrations: used for changes to the ng update migrations.
- ngcc: used for changes to the Angular Compatibility Compiler
- ve: used for changes specific to ViewEngine (legacy compiler/renderer).
- devtools: used for changes in the browser extension.
- none/empty string: useful for test and refactor changes that are done across all packages (e.g. test: add missing unit tests) and for docs changes that are not related to a specific package (e.g. docs: fix typo in tutorial).

Summary

Use the summary field to provide a succinct description of the change:

- use the imperative, present tense: "change" not "changed" nor "changes"
- don't capitalize the first letter
- no dot (.) at the end

Commit Message Body

Just as in the summary, use the imperative, present tense: "fix" not "fixed" nor "fixes".

Explain the motivation for the change in the commit message body. This commit message should explain *why* you are making the change. You can include a comparison of the previous behavior with the new behavior in order to illustrate the impact of the change.

Commit Message Footer

The footer can contain information about breaking changes and deprecations and is also the place to reference GitHub issues, Jira tickets, and other PRs that this commit closes or is related to. For example:

BREAKING CHANGE: <breaking change summary>
<BLANK LINE>
<breaking change description + migration instructions>
<BLANK LINE>
<BLANK LINE>
Fixes #<issue number>

or

DEPRECATED: <what is deprecated>
<BLANK LINE>
<deprecation description + recommended update path>
<BLANK LINE>
<BLANK LINE>
Closes #<pr number>

The Breaking Change section should start with the phrase "BREAKING CHANGE: " followed by a summary of the breaking change, a blank line, and a detailed description of the breaking change that also includes migration instructions.

Similarly, a Deprecation section should start with "DEPRECATED: " followed by a short description of what is deprecated, a blank line, and a detailed description of the deprecation that also mentions the recommended update path.

Language

Typescript

contact

milad.khanmohammadi@gmail.com