Vue 3 projekti malli loomine

Projekti loomine

Javascripti pakettide haldamiseks kasutame pnpm-i. Võrreldes npm-ga hoiab pnpm kettal ruumi kokku, sest see taaskasutab samu pakettide faile, kui neid kasutatakse erinevates projektides sama arvuti peal.

Näites kasutame projekti loomiseks Vite.

Mõned Vite-i eelised vue-cli ees:

- Koodi build on minimaalselt 10 korda kiirem
- Koodis muudatuse tegemine jõuab koheselt brauserisse

Esialgne projekti loomine toimub käsurealt

1. Ava käsurida sobivas kaustas kuhu soovid projekti luua
2. Projekti loomiseks käivitada käsurealt clientapp asemel kirjutada soovitud projekti nimi

pnpm create vite clientapp --template vue-ts

```
PS C:\kool\n4\template> pnpm create vite clientapp --template vue-ts
.../Local/pnpm/store/v3/tmp/dlx-19792    |   +1 +
Packages are hard linked from the content-addressable store to the virtual store.
  Content-addressable store is at: C:\Users\matti\AppData\Local\pnpm\store\v3
  Virtual store is at:             ../../../Users/matti/AppData/Local/pnpm/store/v3/tmp/dlx-19792/node_modules/.pnpm
.../Local/pnpm/store/v3/tmp/dlx-19792    | Progress: resolved 1, reused 0, downloaded 0, added 0

Scaffolding project in C:\kool\n4\template\clientapp...

Done. Now run:

  cd clientapp
  pnpm install
  pnpm run dev
.../Local/pnpm/store/v3/tmp/dlx-19792    | Progress: resolved 1, reused 1, downloaded 0, added 1, done
PS C:\kool\n4\template>
```

3. Powershelli kasutades võib tulla viga „pnpm.psl cannot be loaded because running scripts is disabled on this system.“ Selle lahendamiseks avada Powershell administraatorina ning käivitada käsk

Set-ExecutionPolicy RemoteSigned

4. Avame loodud projekti. Asenda clientapp enda projekti nimega.

cd clientapp

5. Installime paketid

pnpm install

6. Käivitame arendusserveri

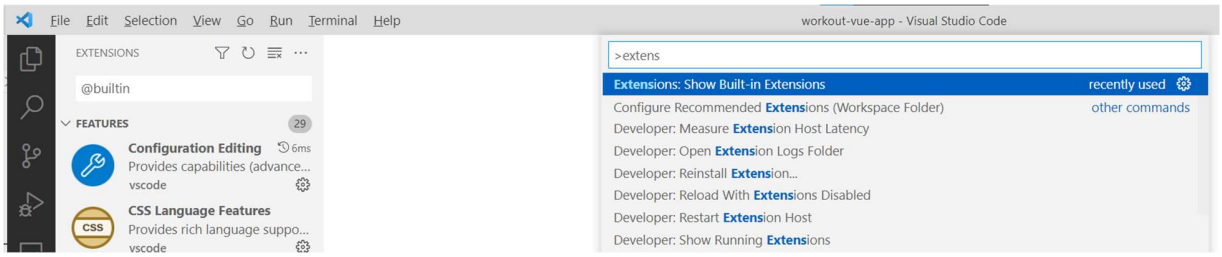
pnpm run dev

7. Kopeerime käsu väljundist aadressi ja avame selle brauseris
8. Arendusserveri seisma jätmiseks terminali aknas Ctrl + C

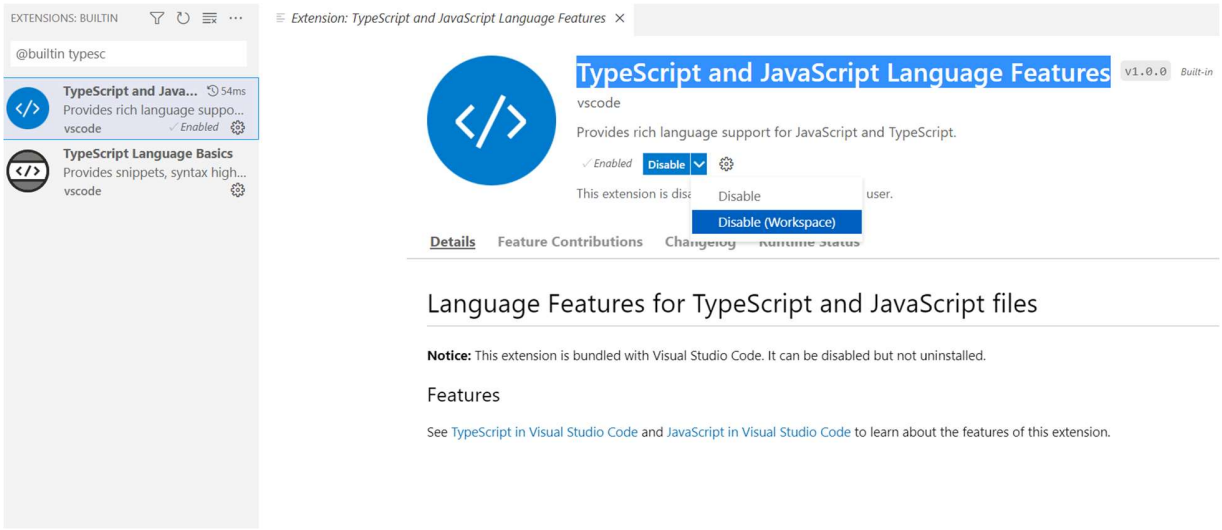
Projekti täiendamine ja Visual Studio Code seadistamine

Lülitame välja VSCode-i sisseehitatud TypeScript toe ja lülitame selle asemel sisse Volar-i TypeScript toe.

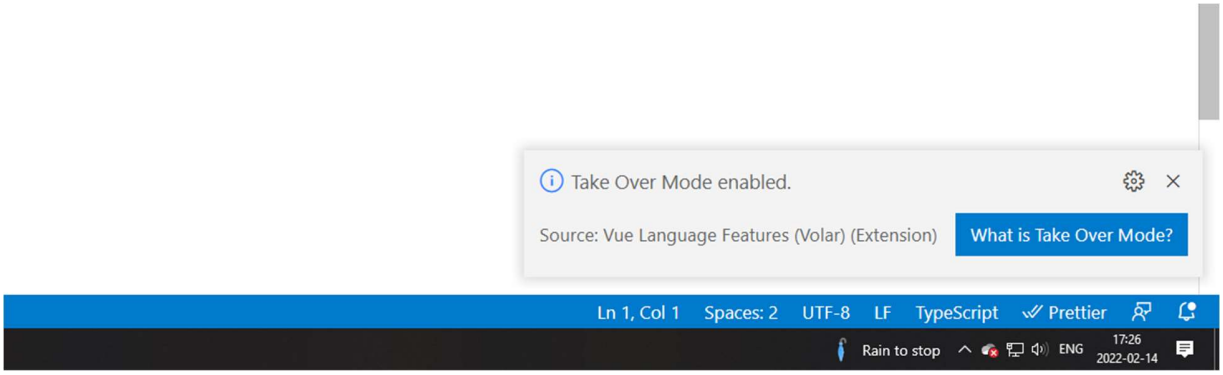
1. Avame VSCode Command palette-I klahviga F1 ja trükime Extensions: Show built-in extensions.



2. Otsime TypeScript and JavaScript Language Features ja lülitame selle välja projekti piires.

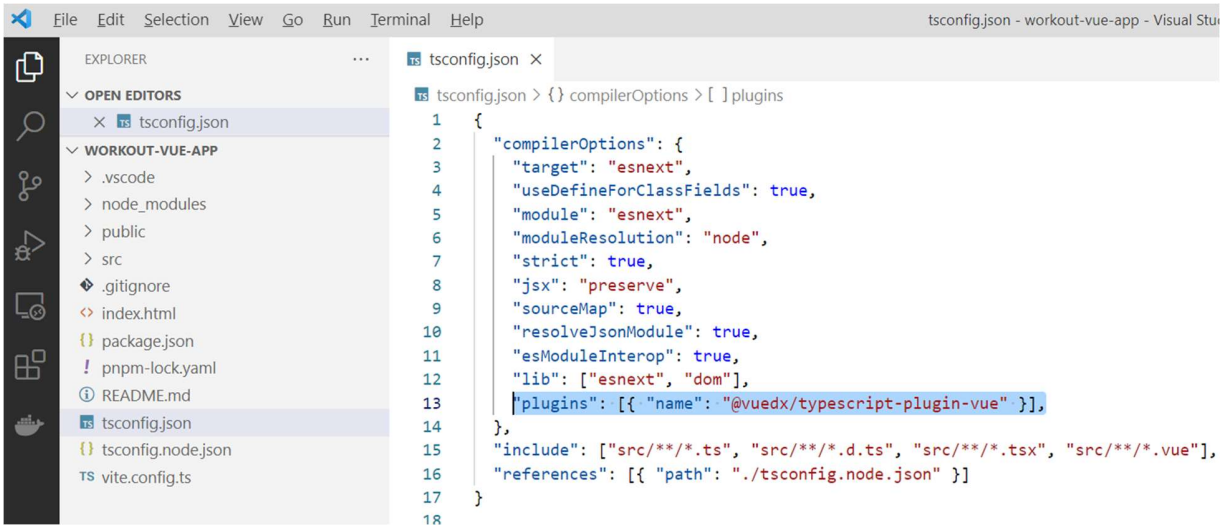


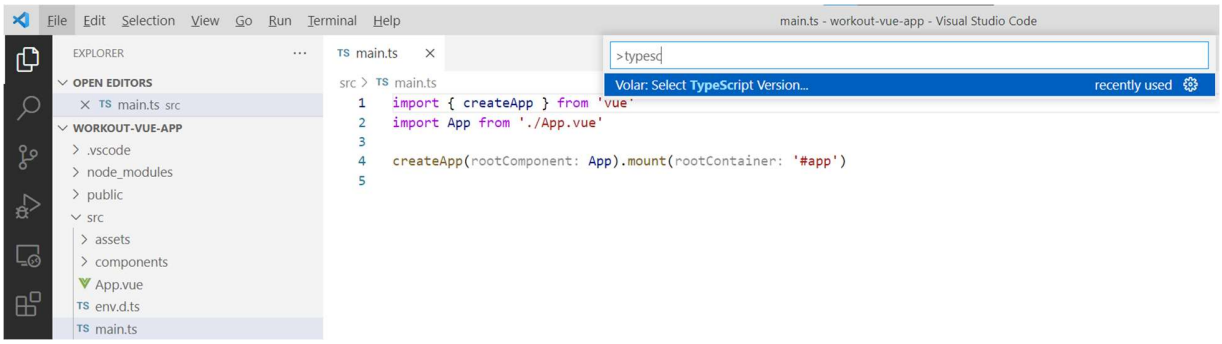
3. VSCode-i alla paremasse nurka ilmub teade.



Typescript-I seadistamine

1. Lisame järgneva rea tsconfig.json faili, "compilerOptions alla
   "plugins": [{ "name": "@vuedx/typescript-plugin-vue" }]



2. Avame src/main.ts faili
3. Avame VSCode Command palette-I klahviga F1 ning kirjutame Select TypeScript Version ja valida Use Workspace Version.
4. 

Soovituslikud lisad koodi vormindamiseks

Prettier

Lisame projektile Prettier paketi.

Eesmärgiks on koodi automaatne vormindamine, et projektis oleks läbivalt kasutusel sama koodistiil.

Käsurealt käivitame

pnpm install -D prettier

Lisame projekti faili .prettierrc.json, mis sisaldab seadistusi.

Näidisprojekti faili sisu on järgnev, kuid võib seadistada vastavalt enda soovidele

```
{
  "printWidth": 80,
  "tabWidth": 2,
  "useTabs": false,
  "semi": true,
  "singleQuote": true,
  "trailingComma": "all"
}
```

Kõik võimalikud valikud on leitavad järgnevalt lehelt https://prettier.io/docs/en/options.html.

Teeme arenduskeskkonna juhendis lisatud Prettier - Code formatter laienduse projekti vaikimisi vormindajaks, selleks lisame .vscode/settings.json faili järgneva rea. Kui faili ei eksisteeri saab avada VSCode-I Command palette-I F1-ga ning trükkida Preferences: Open workspace settings (JSON).

"editor.defaultFormatter": "esbenp.prettier-vscode"

Kui on soovi, et vormindamine tehakse automaatselt kui fail salvestatakse, võib lisada

"editor.formatOnSave": true

ESLint

ESLint eesmärgiks on JavaScript/TypeScript koodi analüüsimine ja sealt potentsiaalsete probleemide tuvastamine.

Kõigepealt installime ESLint paketi

pnpm install -D eslint eslint-config-prettier eslint-plugin-vue

pnpm install -D @typescript-eslint/eslint-plugin @typescript-eslint/parser @typescript-eslint/eslint-plugin @vue/eslint-config-typescript

Lisame projekti faili .eslintrc.rc, mis sisaldab seadistusi.

Näidisprojekti faili sisu on järgnev, võib seadistada vastavalt enda soovidele

```
module.exports = {
 root: true,
 env: {
  node: true,
 },
 extends: [
  'plugin:vue/vue3-recommended',
  'eslint:recommended',
  'prettier',
  'prettier/vue',
  '@vue/typescript',
 ],
 plugins: ['vue'],
};
```

Rohkem seadistamise võimalusi https://eslint.org/docs/user-guide/configuring/.

CSS raamistik – WindiCSS

Selleks, et CSS kirjutamine oleks lihtsam võib kasutusele võtta mõne CSS raamistiku.

Üheks variandiks on WindiCSS, mis on kasutusel ka näidisprojektis.

Raamistiku paketi lisamiseks

```
pnpm i -D vite-plugin-windicss windicss
```

Lisame plugin-I faili vite.config.ts

TS vite.config.ts ×

TS vite.config.ts > ...

```ts
import { defineConfig } from 'vite';
import vue from '@vitejs/plugin-vue';
import WindiCSS from 'vite-plugin-windicss';

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [vue(), WindiCSS()],
});

```

Lisame faili src/main.ts rea

import 'virtual:windi.css'

TS main.ts ×

src > TS main.ts

```ts
import { createApp } from 'vue';
import App from './App.vue';
import 'virtual:windi.css';

createApp(rootComponent: App).mount(rootContainer: '#app');

```

WindiCSS võib muuta Vue 3 vaikimisi kujundust.

primevue

primevue pakub suurt hulka taaskasutatavaid komponente, mis lihtsustavad front end arendust.

Raamistiku paketi lisamiseks

```
pnpm install primevue@^3.17.0 --save
pnpm install primeicons --save
```

Lisame faili src/main.ts read

import PrimeVue from 'primevue/config';

import 'primevue/resources/themes/saga-blue/theme.css'; //theme

import 'primevue/resources/primevue.min.css'; //core css

import 'primeicons/primeicons.css'; //icons

app.use(PrimeVue);

Otseviide src kaustale

Kui kaustade struktuur sügavaks läheb on tüütu kasutada import tegemisel relative path-i. Selle probleemi lahendamiseks on võimalik lisada alias mis viitab otse src kaustale.

Selleks lisada kõigepealt devDependecy

pnpm install -D @types/node

Muuta tuleb ka kahte faili, esiteks tsconfig.json

Seal lisada compilerOptions alla järgnev

"baseUrl": ".",

"paths": { "@/*": ["src/*"] },

"allowSyntheticDefaultImports": true

tsconfig.json

```
{
  "compilerOptions": {
    "target": "esnext",
    "useDefineForClassFields": true,
    "module": "esnext",
    "moduleResolution": "node",
    "strict": true,
    "jsx": "preserve",
    "sourceMap": true,
    "resolveJsonModule": true,
    "esModuleInterop": true,
    "lib": ["esnext", "dom"],
    "plugins": [{ "name": "@vuedx/typescript-plugin-vue" }],
    "baseUrl": ".",
    "paths": { "@/*": ["src/*"] },
    "allowSyntheticDefaultImports": true
  },
  "include": ["src/**/*.ts", "src/**/*.d.ts", "src/**/*.tsx", "src/**/*.vue"],
  "references": [{ "path": "./tsconfig.node.json" }]
}
```

Teine fail mida muuta tuleb on vite.config.ts

Seal lisada import lausena import * as path from 'path' ja export default defineConfig({ alla

resolve: {

alias: { '@': path.resolve(__dirname, 'src') },

},

vite.config.ts

```
import { defineConfig } from 'vite';
import vue from '@vitejs/plugin-vue';
import WindiCSS from 'vite-plugin-windicss';
import * as path from 'path';

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [vue(), WindiCSS()],
  resolve: {
    alias: { '@': path.resolve(__dirname, 'src') },
  },
});
```