

Template-генератор для методологии FSD

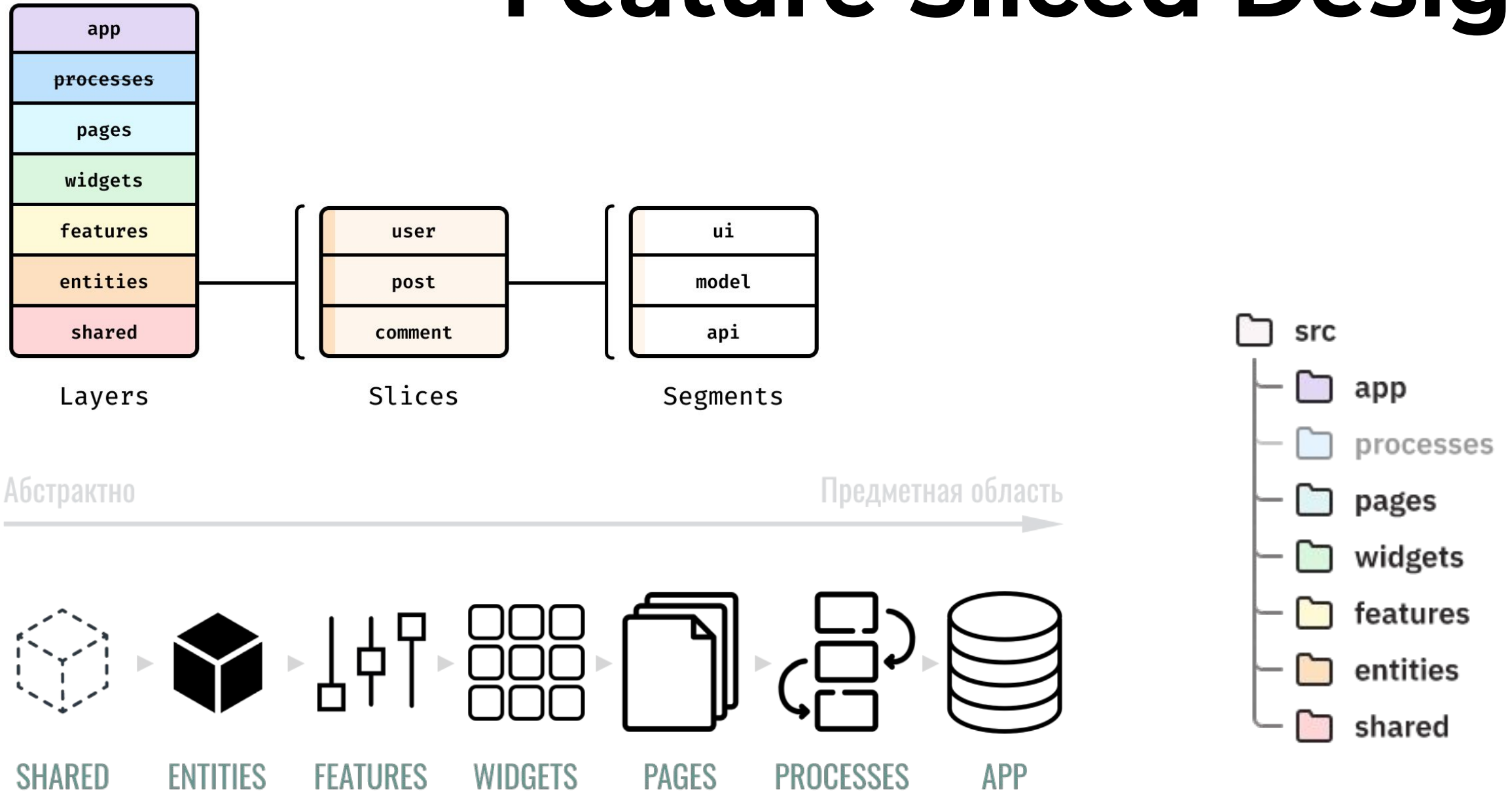
ДВФУ, «Прикладная информатика в компьютерном дизайне»,
2023г.

Рузин Михаил
Баздуков Валентин
Ли Дмитрий

Преподаватель: Кленина Надежда Викторовна



Feature Sliced Design



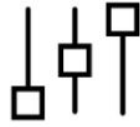
Feature Sliced Design



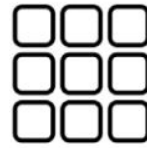
SHARED



ENTITIES



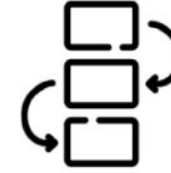
FEATURES



WIDGETS



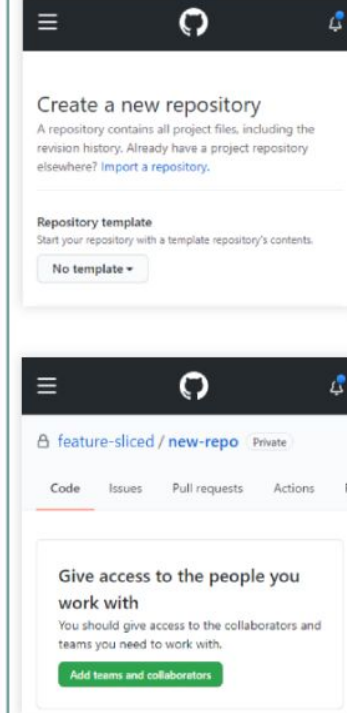
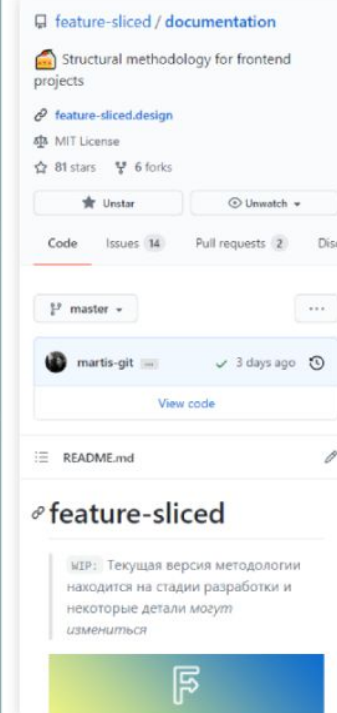
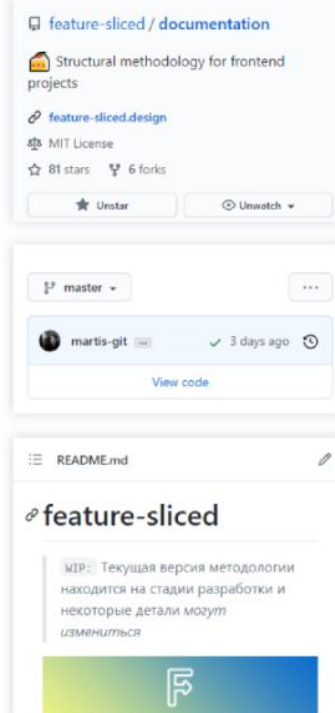
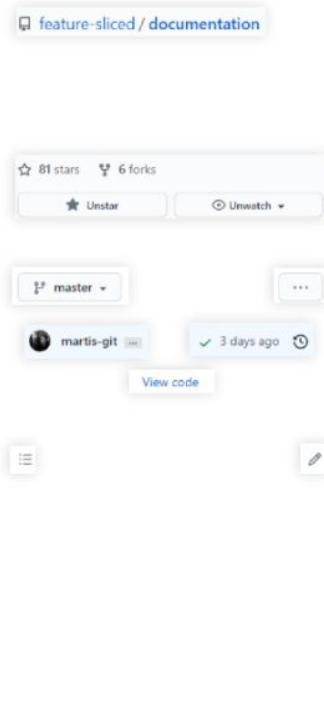
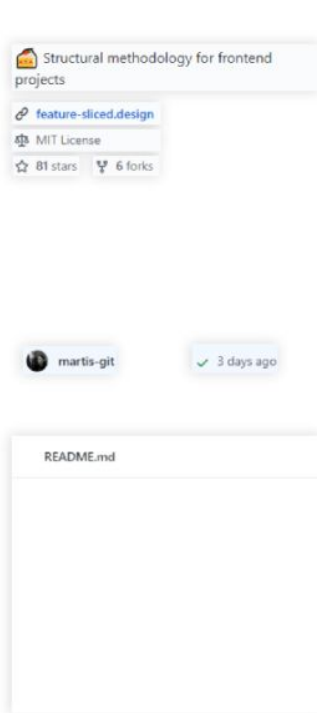
PAGES



PROCESSES



APP



- + withRouter
- + withStore
- + withGlobalStyles
- + withUIKitConfig
- + ...

Проблема

- Методология накладывает ограничения на структуру проекта
- Из-за этого возникает большое количество шаблонного кода
- Написание шаблонного кода можно автоматизировать
- Нет решений, которые бы позволяли покрывать особенности FSD из коробки и при этом распространялись бы открыто без привязки к средам разработки

Название	Политика распространения	Отсутствие завязанности на конкретном редакторе	Распространение через prtm	Применимость к фронтенд-разработке	Покрытие задач, связанных с FSD	Открытый исходный код	Готовность к использованию «из коробки»
JetBrains templates	+	-	-	+	+	-	+
Snippets Vs-code	+	-	-	+	-	+	+
nest-cli	+	+	+	-	-	+	+
codesmith	-	+	-	+	-	-	+
vgent	+	+	+	+	-	+	+
generate-react-cli	+	+	+	+	-	+	+
hygen	+	+	+	+	+	+	-
yeoman	+	+	+	+	+	+	-
feature-sliced-design cli	+	+	+	+	+	+	-
Emmet	+	+	-	+	-	+	+
Suipta	+	+	+	+	+	+	+

Шаблонизатор с консольным интерфейсом

2 функции: генератор слайса и генератор компонента в сегменте

Распространение через npm

Suipta



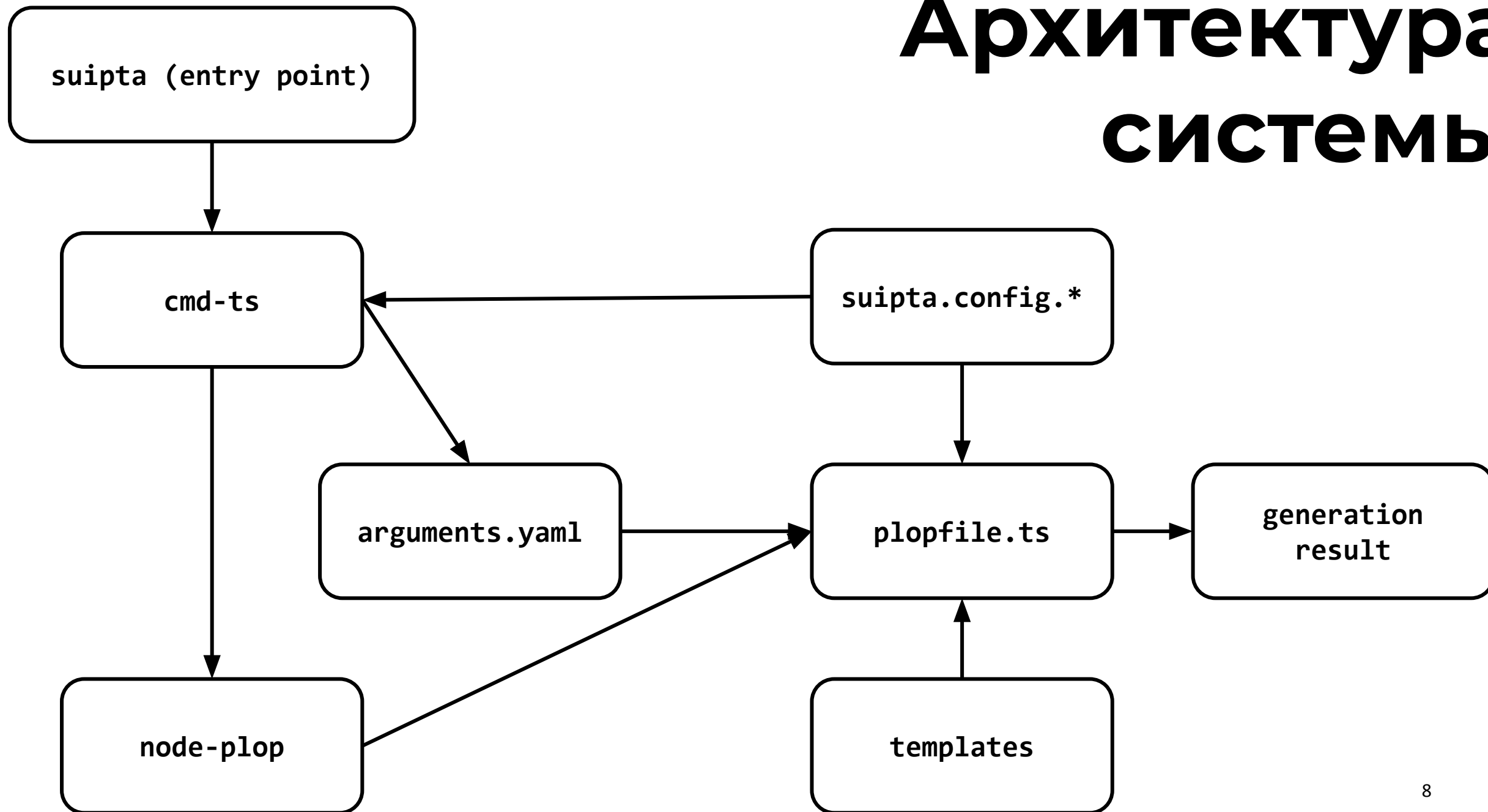
handlebars



GitHub Actions




Архитектура системы



Формат команды

```
suipta [-h | -help] GENERATOR [LAYER] \  
[SLICE | SEGMENT [COMPONENT]] [-ui] [-m | -model] \  
[-l | -language]
```

Формат конфига



```
/**
 * @type {import('suipta').SuiptaConfig}
 */
export default {
  layers: ['entities', 'features', 'widgets', 'pages'],
  segmentLayers: ['shared'],
  segment: ['lib', 'ui'],
  templatesDir: './templates',
  lang: 'ts',
  rootDir: './src'
}
```

Формат шаблонов

```
entities/ui.tsx

import s from './s.module.scss'
import { {{pascalCase slice }}Props } from '../types'

export const {{ pascalCase slice }} = (props: {{ pascalCase slice }}Props) => {
  return <React.</>
}
```

Тестирование

одного из сценариев использования

```
it('with ui argument = react and model = effector', async () => {
  const slice = 'with-effector-react'
  await suiptaHandler({
    generator: 'slice',
    layer: 'entities',
    slice,
    ui: 'react',
    model: 'effector',
  })

  const slicePath = path.join(generationPath, 'entities', slice)

  expect(fs.existsSync(slicePath)).toBeTruthy()
  testEntities(slice)
})
```

```
export function testEntities(name: string) {
  const slicePath = path.join(generationPath, 'entities', name)
  const resultPath = path.join(resultsPath, 'entities', name)
  const pathsToCheck = [
    path.join('ui', 'index.tsx'),
    path.join('ui', 's.module.scss'),
    path.join('model', 'index.ts'),
    'types.ts',
    'index.ts',
  ]
  pathsToCheck.forEach(item => {
    expect(fs.readFileSync(path.join(slicePath, item)).toString()).toMatch(
      fs.readFileSync(path.join(resultPath, item)).toString()
    )
  })
}
```

Результаты

- Npm пакет @strwatcher/suipta
- Репозиторий на github с документацией и исходным кодом
- Автоматизированный пайплайн выпуска релизов