```
// /**
//  * @function
//  * @param {express.Request} req
//  * @param {express.Response} res
//  * @param {express.NextFunction} next
//  * @returns {Promise < void >}
//  */
// async function findById(req, res, next) {
//     try {
//         const { error } = UserValidation.findById(req.params);

//         if (error) {
//             throw new ValidationError(error.details);
//         }

//         const user = await UserService.findById(req.params.id);

//         return res.status(200).json({
//             data: user,
//         });
//     } catch (error) {
//         if (error instanceof ValidationError) {
//             return res.status(422).json({
//                 error: error.name,
//                 details: error.message,
//             });
//         }

//         res.status(500).json({
//             message: error.name,
//             details: error.message,
//         });

//         return next(error);
//     }
// }

/**
```

```
57          * Renders user page with popup-update.
58          * @name /users
59          * @function
60          * @inner
61          * @param {string} path - Express path
62          * @param {callback} middleware - Express middleware.
63          */
64         app.post('/users/update', async (req, res, next) => {
65             try {
66                 const users = await UserService.findAll();
67
68                 res.render('users', {users: users, popup: 'update', id: req.body.id});
69             } catch (error) {
70                 res.render('users', {users: null, popup: null});
71
72                 next(error);
73             }
74         });
75
76         /**
77          * Forwards any requests to the /v1/users URI to UserRouter.
78          * @name /v1/users
79          * @function
80          * @inner
81          * @param {string} path - Express path
82          * @param {callback} middleware - Express middleware.
83          */
84         app.use('/v1/users', UserRouter);
85
```

```
async function create(req, res, next) {
    try {
        const { error } = UserValidation.create(req.body);

        if (error) {
            throw new ValidationError(error.details);
        }

        const user = await UserService.create(req.body);

        res.redirect('/v1/users/');
    } catch (error) {
        if (error instanceof ValidationError) {
            return res.status(422).render('error.ejs', {
                message: error.name,
                details: error.message,
            });
        }

        res.status(500).render('error.ejs', {
            message: error.name,
            details: error.message,
        });

        return next(error);
    }
}
```

```
{
    message: "E_MISSING_OR_INVALID_PARAMS",
  - details: [
      - {
            message: ""fullName" is not allowed to be empty",
          - path: [
                "fullName"
            ],
            type: "string.empty",
          - context: {
                label: "fullName",
                value: "",
                key: "fullName"
            }
        }
    ]
}
```

```
try {
  const dataValidation = { ...req.params, ...req.body };
  const { error } = UserValidation.updateById(dataValidation);
  if (error) {
    throw new ValidationError(error.details);
  }
  await UserService.updateById(req.params.id, req.body);
  req.flash('info', 'fullName updated');
  return res.status(200).redirect('/v1/users/');
} catch (error) {
  if (error instanceof ValidationError) {
    return res.status(422).json({
      message: error.name,
      details: error.message,
    });
```

eslint no-console

```
} catch (error) {
    console.error(error);
    res.status(500).render({
        error: '500',
        message: error.message[0].message,
    });

    next(error);
}
```

```
if (error instanceof ValidationError) {
    return res.status(422).render('errors/validError.ejs', {
        csrfToken: req.csrfToken(),
        method: 'put',
        name: error.name,
        message: error.message[0].message,
        id: req.body.id,
    });
}

res.status(500).render('errors/validError.ejs', {
    csrfToken: req.csrfToken(),
    method: 'put',
    name: error.name,
    message: error.message,
    id: req.body.id,
});
```

Я заметил :)

feat: 🎸 create view

feat: 🎸 new project

feat: 🎸 new project

feat: 🎸 new project

feat: 🎸 create view

feat: 🎸 add new project

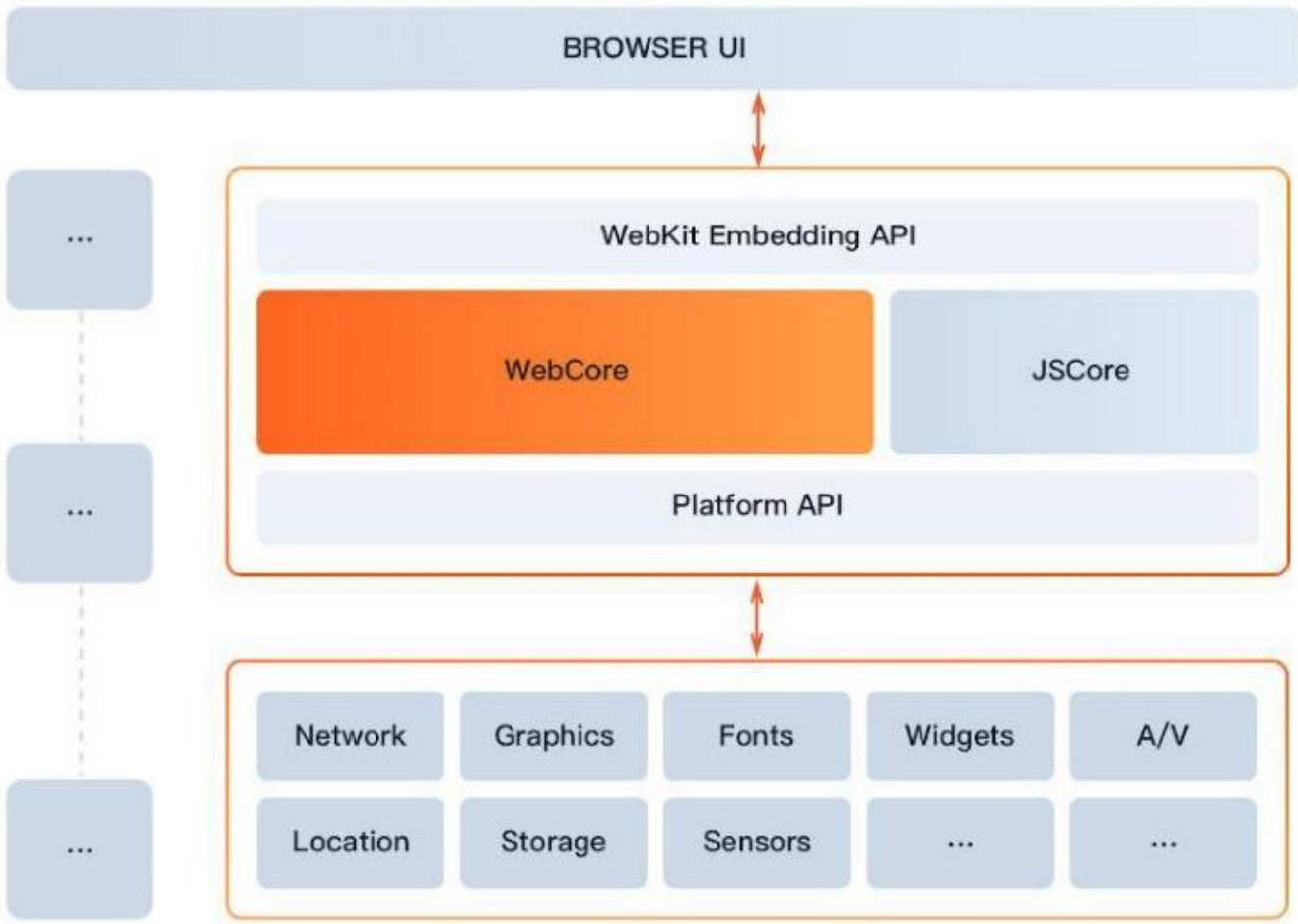
feat: 🎸 create view

feat: 🎸 create view

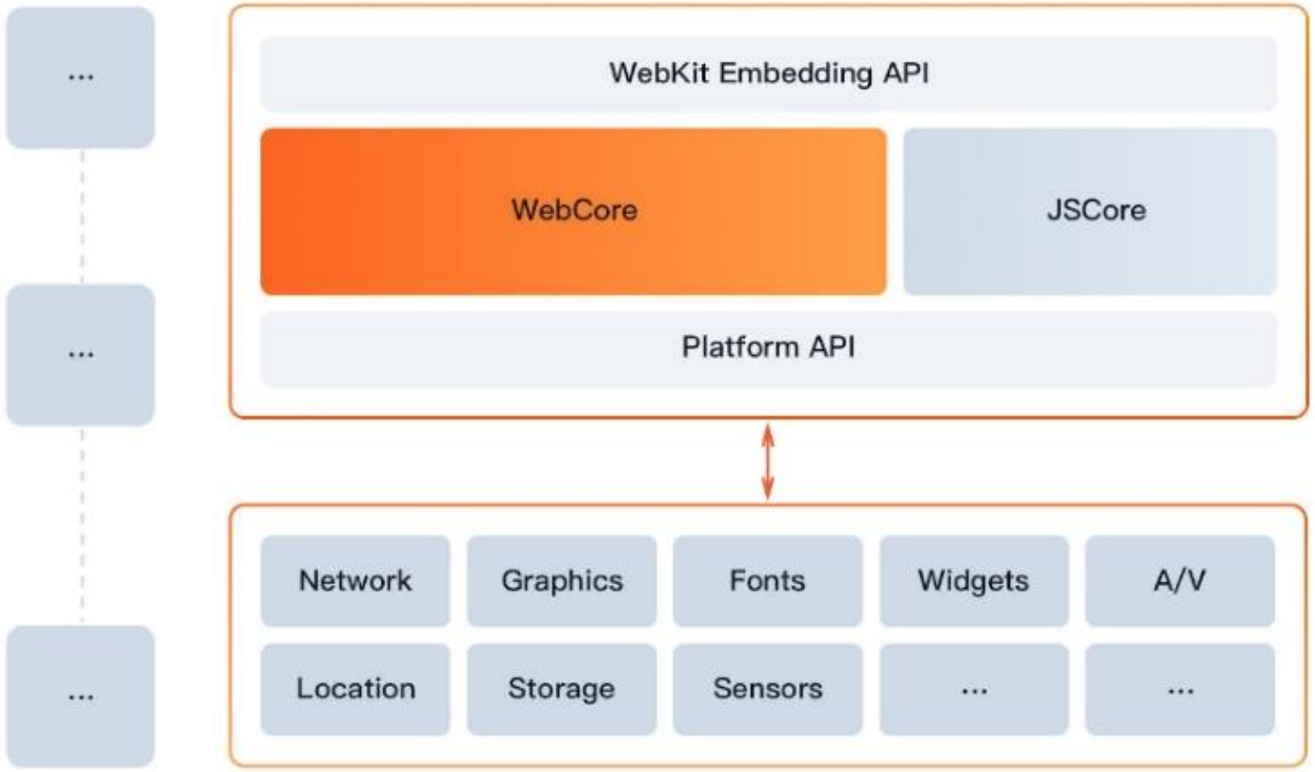
FOREACH

headless browser

Обычный браузер



Безголовый браузер



Chrome

Firefox

PhantomJS

Splash

TrifleJS

Test automation

Screenshot of web pages

Scraping

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

Save List of emails to MondoDB collection ‘GrabbingEmails’

Create screenshot from users table and upload to google drive Save public link to file in collection ‘Screenshots’