Rust

동계 모각소
러스트 학습

이웅희

학습 계획

1~3주차

교재 - 기본 문법 학습, 챕터 1~4

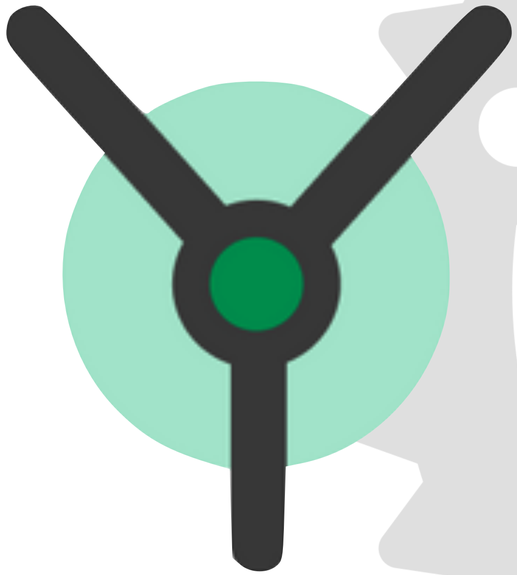
4주차

Yew.rs - Front-end

5주차

Rocket.rs - Back-end

개요 - Yew 소개



Yew

WASM를 활용한 Rust의 Front-end 프레임워크

Javascript의 React와 유사한 V-DOM 기반 렌더링

trunk를 통해 WASM로 번들 및 빌드 진행

개요 - Yew 소개

```rust
use yew::prelude::*;

#[function_component(App)]
pub fn app() -> Html {
    html! {
        <main>
            <img class="logo" src="https://yew.rs/img/logo.png" alt="Yew logo" />
            <h1>{ "Hello World!" }</h1>
            <span class="subtitle">{ "from Yew with " }<i class="heart" /></span>
        </main>
    }
}
```

함수형 컴포넌트

HTML 템플릿

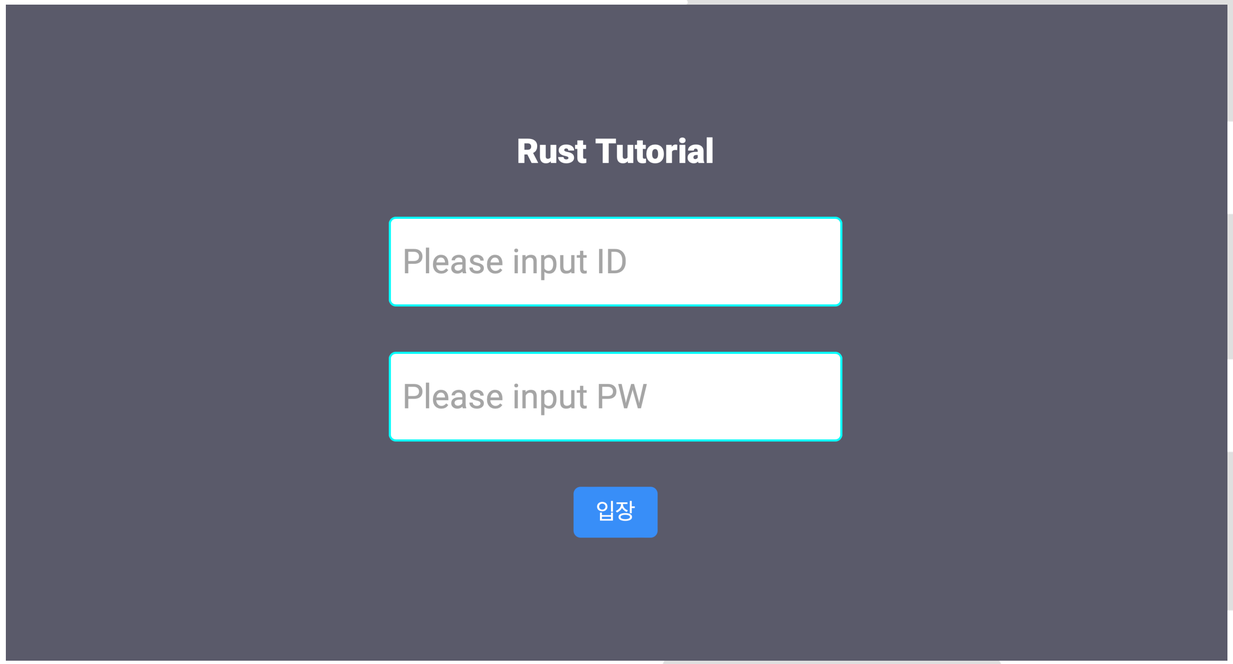
Ex) app.rs

```rust
mod app;

use app::App;

fn main() {
    yew::Renderer::<App>::new().render();
}
```

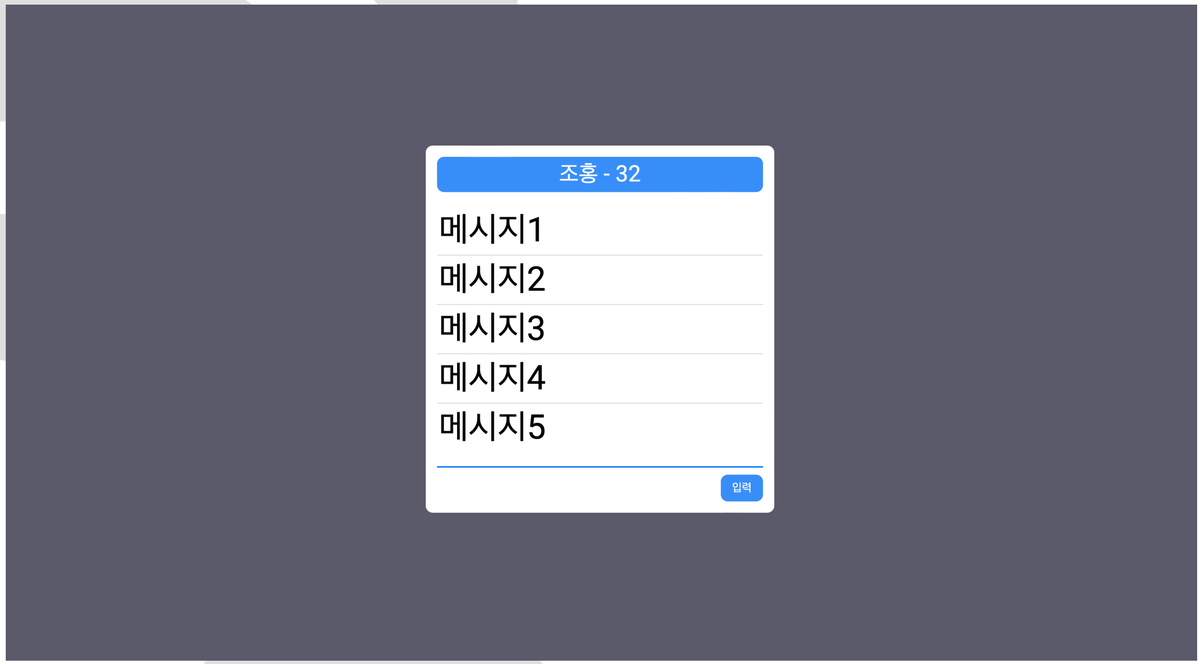
컴포넌트 임포트

Ex) main.rs

개요 - 구현 결과 화면



화면 1. <로그인>



화면 2. <메시지 입력>

화면 1. <로그인>

```
<form class={classes!("form")} {onsubmit}>
    <div class={classes!("title")}>{"Rust Tutorial"}</div>
    <input class={classes!("input")} type="text" placeholder="Please input ID" oninput={oninput_id} />
    <input class={classes!("input")} type="password" placeholder="Please input PW" oninput={oninput_pw} />
    <button class={classes!("enter")}>{"입장"}</button>
</form>
```

HTML 템플릿 작성

classes! 매크로를 통해 CSS 적용

```
let id = use_state(|| "".to_string());
let pw = use_state(|| "".to_string());

let oninput_id = Callback::from({
    let id = id.clone();
    move |input_event: InputEvent| {
        let target: HtmlInputElement = input_event.target_dyn_into().unwrap();

        id.set(target.value());
    }
});
let oninput_pw = Callback::from({
    let pw = pw.clone();
    move |input_event: InputEvent| {
        let target: HtmlInputElement = input_event.target_dyn_into().unwrap();

        pw.set(target.value());
    }
});
```

use_state 함수로 상태 관리

콜백 함수 및 클로저를 활용하여 이벤트 관리

clone과 setter를 활용한 불변성 유지

화면 1. <로그인>

```rust
#[derive(Default, Clone, PartialEq, Eq)]
struct User {
    pub username: String,
    pub user_number: i32,
}
```

struct를 활용해 구조체 타입 지정

Default 등 트레이트 불러와 사용

```rust
let user = use_state(|| User::default());
let cloned_user = user.clone();

let onsubmit = Callback::from({
    let id = id.clone();
    let pw = pw.clone();

    move |form_event: SubmitEvent| {
        form_event.prevent_default();

        log!("Submit!".to_string());

        log!(id.to_string());
        log!(pw.to_string());

        // HTTP GET
        let mut user = cloned_user.deref().clone();
        user.username = id.to_string();
        user.user_number = 32;

        cloned_user.set(user);
    }
});
```

use_state 함수로 상태 관리

콜백 함수 및 클로저를 활용하여 이벤트 관리

mut 키워드를 통해 가변 참조

다음 주차에 API 코드 구현 예정

화면 2. <메시지 입력>

```
html! {
    <div class={classes!("main")}>
    if user.deref().clone().eq(&User::default()) {…
        } else {…
    </div>
} html!
```

If-else 블록으로 조건부 렌더링

```
<div class={classes!("container")}>
    <div class={classes!("container__title")}>{"조홍 - 32"}</div>
    <ul class={classes!("container__list")}>
        {
            msg.into_iter().map(|m| {
                html! {<li key={m} class={classes!("container__item")}>{m}</li>}
        }).collect::<Html>()
    }
    </ul>
    <form class={classes!("container__form")}>
        <input class={classes!("container__input")} type="text" />
        <button class={classes!("container__button")}>{"입력"}</button>
    </form>
</div>
```

고차함수 map을 활용해 리스트 렌더링

다음 주차에 API 코드 구현 예정

```
<link data-trunk rel="rust" />
<link data-trunk rel="css" href="style.css" />
```

헤더 태그에 data- 어트리뷰트 추가

부록 - 프로젝트 구조

```
.
├── Cargo.lock
├── Cargo.toml
├── dist
│   ├── index.html
│   ├── style-23ee3d6e68b6e526.css
│   ├── yew-app-716fefe99dde90e2.js
│   └── yew-app-716fefe99dde90e2_bg.wasm
├── index.html
├── src
│   └── main.rs
└── style.css

3 directories, 9 files
```

개요 - Rocket 소개

Rocket

Rust의 Back-end 프레임워크

다른 언어의 웹 프레임워크와 사용이 비슷하여 입문에 용이

Rust의 장점을 적극 반영하여 안전하고 확장성 높은 개발 가능

개요 - Rocket 소개

```rust
#[macro_use]
extern crate rocket;

#[get("/")]
fn index() -> &'static str {
    "Hello, world!"
}

#[launch]
fn rocket() -> _ {
    rocket::build().mount("/", routes![index])
}
```

Rocket 임포트

라우터 설정

앱 실행

Ex) main.rs

개요 - 구현 API 계획

GET [‘/‘] - Health Check

POST [‘/auth‘] - 로그인

GET [‘/<id>‘] - 메시지 읽어오기

POST [‘/msg’] - 메시지 쓰기

GET [‘/‘] - Health Check

```rust
#[get("/")]
fn index() -> String {
    let now = SystemTime::now()
        .duration_since(SystemTime::UNIX_EPOCH)
        .unwrap()
        .as_millis()
        .to_string();

    format!("Now: {}", now)
}
```

현재 시각을 출력하여 서버 상태 확인

결과: `Now: 1677188210191`

POST [‘/auth‘] - 로그인

```rust
#[derive(Database)]
#[database("db")]
struct Db(sqlx::SqlitePool);

async fn run_migrations(rocket: Rocket<Build>) -> fairing::Result {
    match Db::fetch(&rocket) {
        Some(db) => match sqlx::migrate!("db/migrations").run(&**db).await {
            Ok(_) => Ok(rocket),
            Err(e) => {
                error!("Failed to initialize SQLx database: {}", e);
                Err(rocket)
            }
        },
        None => Err(rocket),
    }
}
```

```rust
fn stage() -> AdHoc {
    AdHoc::on_ignite("SQLx Stage", |rocket| async {
        rocket
            .attach(Db::init())
            .attach(AdHoc::try_on_ignite("SQLx Migrations", run_migrations))
    })
}
```

SQLite 설정 및 migration 적용

POST [‘/auth‘] - 로그인

```rust
#[derive(Deserialize, Serialize, Clone)]
struct User {
    #[serde(skip_deserializing, skip_serializing_if = "Option::is_none")]
    id: Option<i64>,
    username: String,
}

#[post("/auth", data = "<user>")]
async fn hello(mut db: Connection<Db>, user: Json<User>) -> Result<Json<User>> {
    let result = sqlx::query!("INSERT INTO users (username) VALUES (?)", user.username,)
        .execute(&mut *db)
        .await?;

    Ok(Json(User {
        id: Some(result.last_insert_rowid()),
        username: user.username.clone(),
    }))
}
```

로그인 로직 구현

결과:

id	username
1	unghee
2	unghee
3	unghee
4	unghee


```
{
    "id": 4,
    "username": "unghee"
}
```

GET [‘/<id>‘] - 메시지 읽어오기

```rust
#[get("/<id>")]
async fn read(mut db: Connection<Db>, id: i64) -> Option<Json<Msg>> {
    sqlx::query!(
        "SELECT id, user_id, text FROM messages WHERE user_id = ?",
        id
    )
    .fetch_one(&mut *db)
    .map_ok(|r| {
        Json(Msg {
            id: Some(r.id),
            user_id: r.user_id,
            text: r.text,
        })
    })
    .await
    .ok()
}
```

메시지 읽기 로직 구현

결과:

```json
{
    "id": 1,
    "user_id": 3,
    "text": "Hello World!"
}
```

POST [‘/msg’] - 메시지 쓰기

```rust
#[derive(Deserialize, Serialize, Clone)]
struct Msg {
    #[serde(skip_deserializing, skip_serializing_if = "Option::is_none")]
    id: Option<i64>,
    user_id: i64,
    text: String,
}
#[post("/msg", data = "<msg>")]
async fn write_msg(mut db: Connection<Db>, msg: Json<Msg>) -> Result<Json<Msg>> {
    let result = sqlx::query!(
        "INSERT INTO messages (user_id, text) VALUES (?, ?)",
        msg.user_id,
        msg.text,
    )
    .execute(&mut *db)
    .await?;

    Ok(Json(Msg {
        id: Some(result.last_insert_rowid()),
        user_id: msg.user_id.clone(),
        text: msg.text.clone(),
    }))
}
```

메시지 입력 로직 구현

결과:

id	user_id	text
1	3	Hello World!


```
{
    "id": 1,
    "user_id": 3,
    "text": "Hello World!"
}
```

부록 - 프로젝트 구조

```
.
├── Cargo.lock
├── Cargo.toml
├── Rocket.toml
├── db
│   ├── db.sqlite
│   ├── db.sqlite-shm
│   ├── db.sqlite-wal
│   └── migrations
│       ├── 20230223230641_initial-tables.sql
│       └── db.sql
├── sqlx-data.json
└── src
    └── main.rs

4 directories, 10 files
```