Exploring Web Development with Rust

Whoami

Nikita (bshn) Puzankov

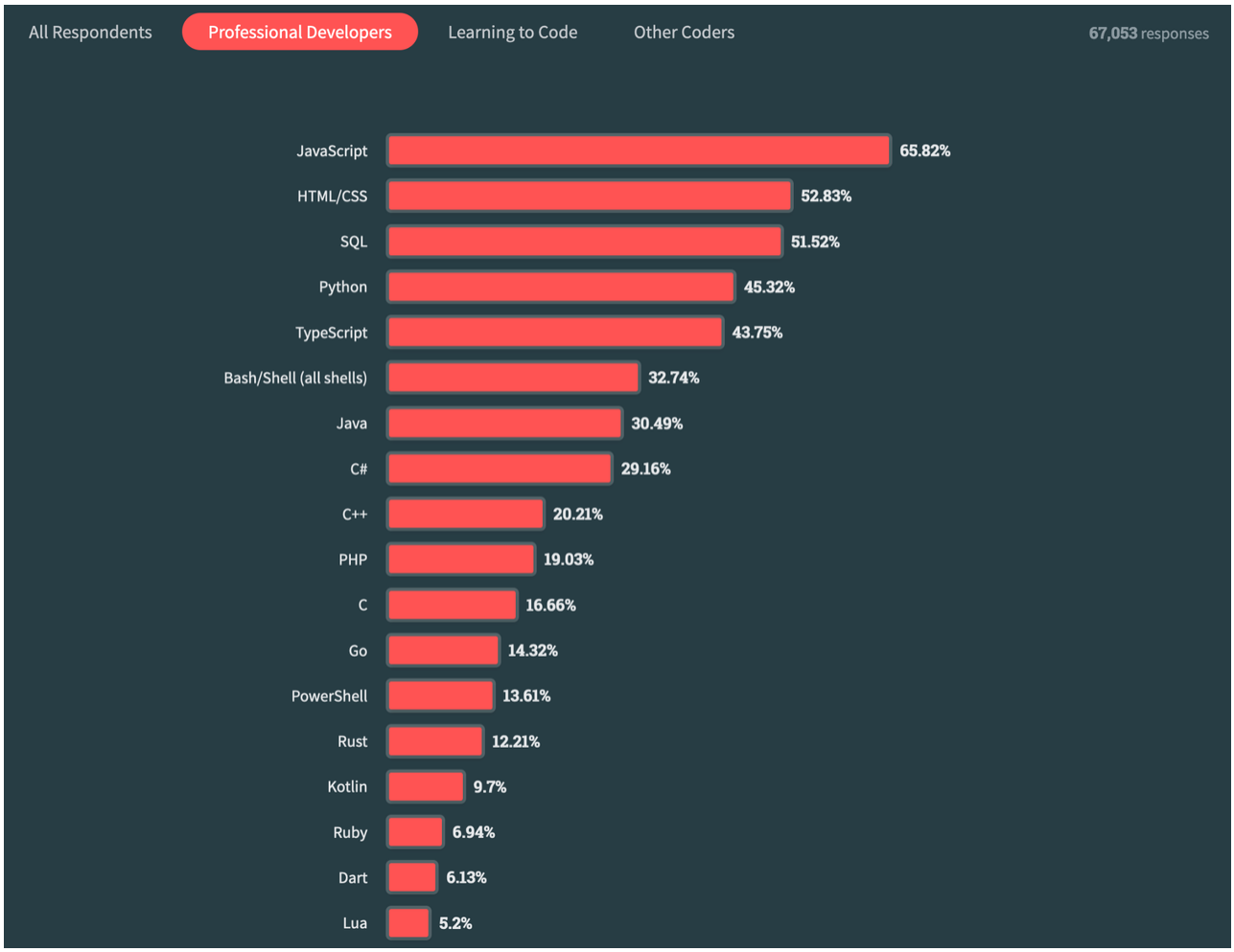
Rust Tbilisi kickoff
18.11.2023

Agenda

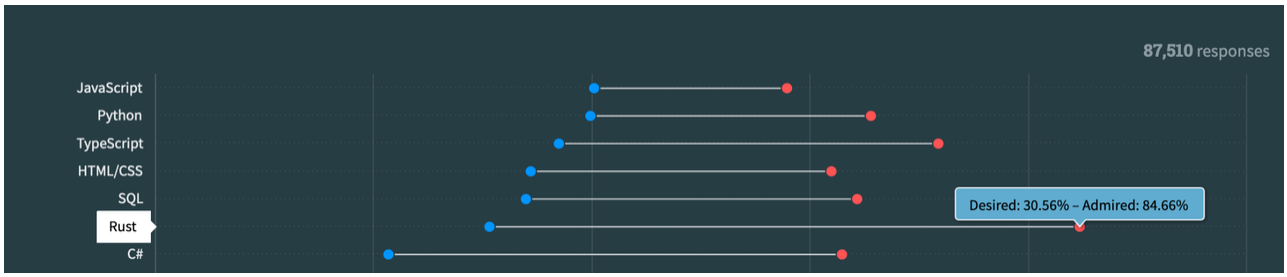
- State of Rust 2023
- Web Development Landscape
- Personal Cookbook

State of Rust 2023

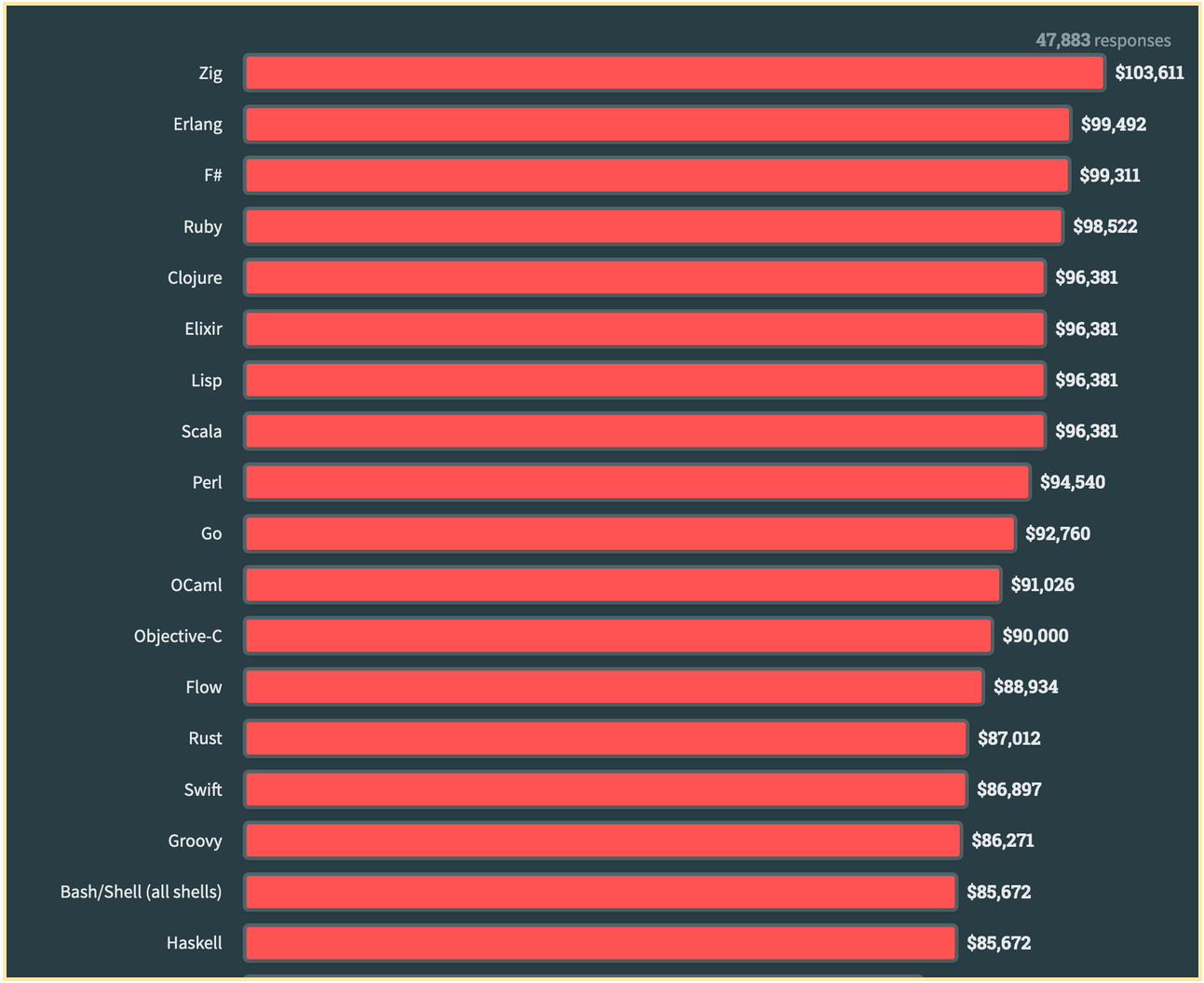
Stack Overflow Survey



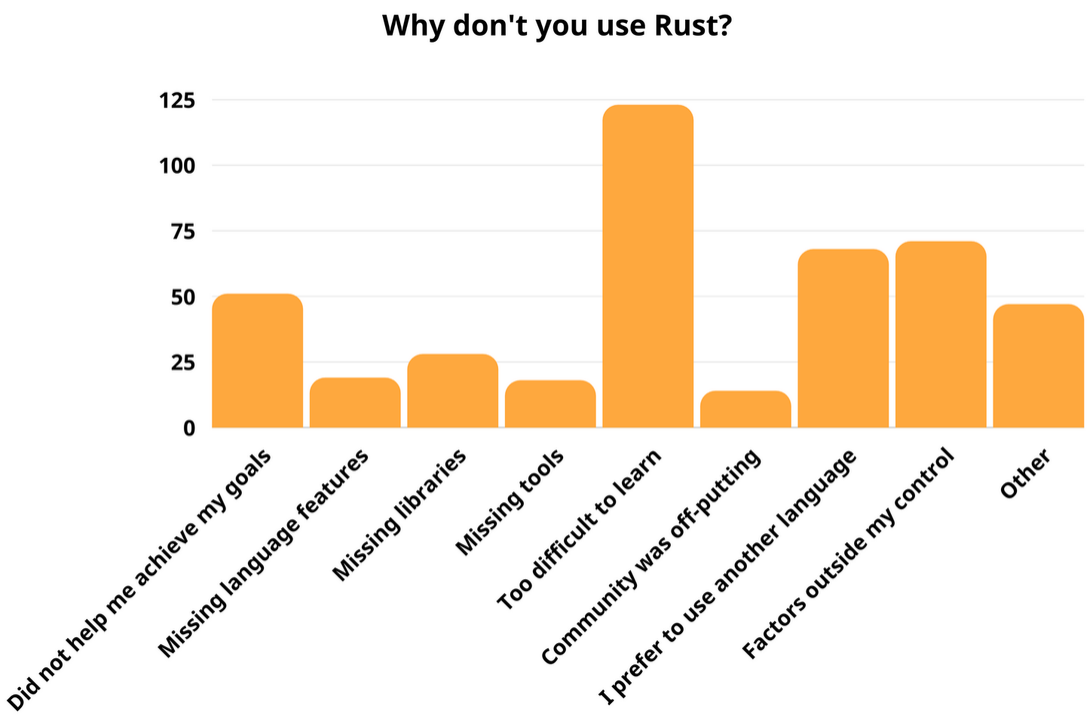
> *Rust is the most admired language, more than 80% of developers that use it want to use it again next year.*



$$$


47,883 responses
Zig
$103,611
Erlang
$99,492
F#
$99,311
Ruby
$98,522
Clojure
$96,381
Elixir
$96,381
Lisp
$96,381
Scala
$96,381
Perl
$94,540
Go
$92,760
OCaml
$91,026
Objective-C
$90,000
Flow
$88,934
Rust
$87,012
Swift
$86,897
Groovy
$86,271
Bash/Shell (all shells)
$85,672
Haskell
$85,672

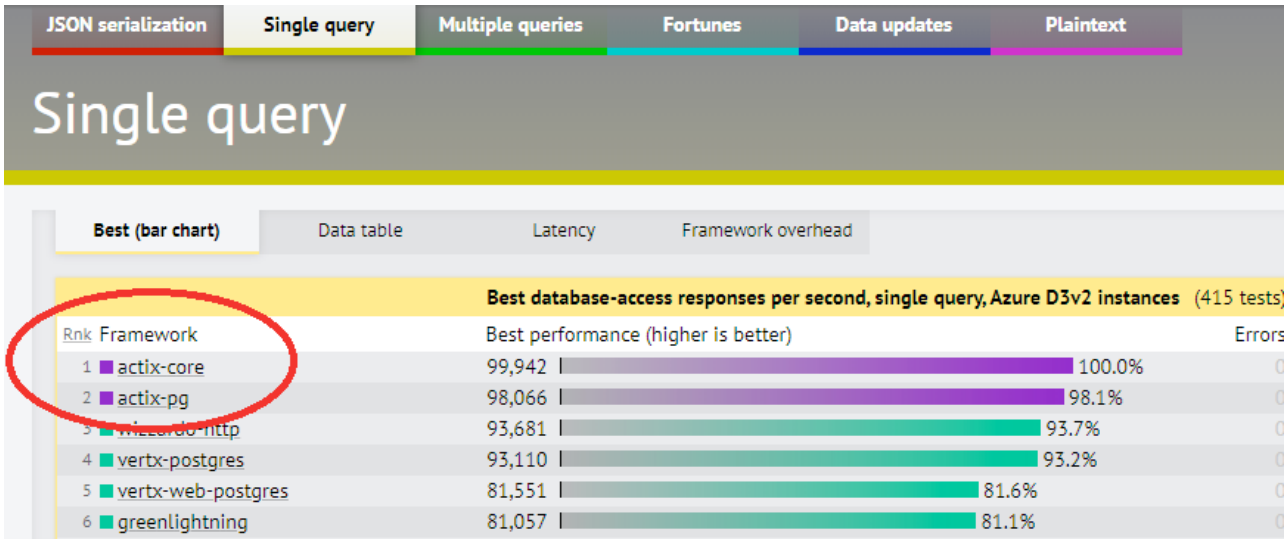
Blockers ⛔⛔⛔

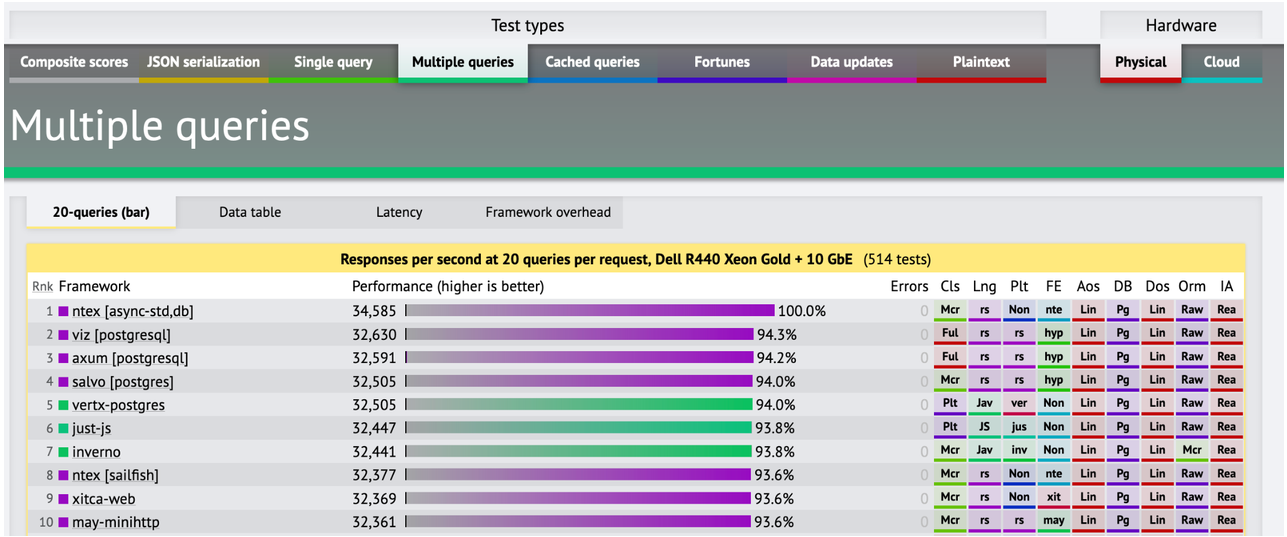


Web Development Landscape

2015-2016

Benchmark hype





Features	Actix	Rocket	Axum	warp
Async/await support	✅	✅	✅	✅
Middleware	✅	✅	✅	✅
WebSockets support	✅	✅	✅	✅
Concurrerncy & Performance	✅	🚫	✅	🚫
Cookie and session	✅	✅	✅	✅
Community growth size	✅	✅	✅	✅

- Async Runtime
  - Tokio
  - async-std
  - glommio
  - ...
- ORM vs compile-time vs raw SQL
  - sqlx
  - ormx/SeaORM
  - Rust-Postgres
  - ...
  - NoSQL
- API
  - gRPC
  - REST
  - GraphQL
  - ...
- Authentication and Authorization

- Observability
- ...

Personal Cookbook

Cargo Workspace Layout

```
├── CONTRIBUTING.md
├── Cargo.lock
├── Cargo.toml
├── Dockerfile
├── README.md
├── cliff.toml
├── docker-compose.yml
├── libraries
│   ├── blockchain
│   ├── database
│   ├── graphql
│   ├── grpc
│   ├── observability
│   ├── rest
│   └── wrappers
├── rust-toolchain.toml
├── services
│   ├── domain
│   └── infrastructure
├── tests
└── test.env
```

Common dependencies

```
[workspace.package]
version = "4.2.8"
```

```
edition = "2021"


[workspace.dependencies]
async-graphql = { version = "5.0.10", features = ["smol_str", "decimal", "uuid", "url"] }
async-graphql-axum = "5.0.10"
axum = { version = "0.6.16", features = ["headers", "tracing", "macros"] }
axum-test = "9.1.1"
derive-new = "0.5"
derive_more =  { version = "1.0.0-beta.3", features = ["full"] }
dotenv = "0.15"
educe = "0.4.22"
envy = "0.4.2"
eyre = "0.6.8"
jwt-authorizer = { git = "https://gitlab.com/oss47/jwt-authorizer.git" }
...
```

Crate layout

```
./services/infrastructure/accounts
├── Cargo.toml
├── migrations
└── src
    ├── api.rs
    ├── configuration.rs
    ├── db.rs
    ├── graphql_api.rs
    ├── lib.rs
    ├── main.rs
```

```
    └── rest_api.rs
```

Stack

Runtime

tokio.rs

```
#[tokio::main]
async fn main() -> Result<()> {
...
#[tokio::test]
async fn end_to_end() {
...
```

HTTP Server

axum

`main.rs`

```
    Server::bind(&configuration.rest.address)
        .serve(rest_api::init().await?.into_make_service())
        .await?;
```

`rest_api.rs`

```
pub async fn init() -> Result<Router> {
    let configuration = Configuration::from_env()?;
    let schema = graphql_api::schema().await?;
    Ok(Router::new()
        .route(
            GRAPHQL_ROOT_PATH,
            routing::post(graphql::handlers::graphql_handler::
<Query, Mutation, EmptySubscription>),
        )
        .route_service(GRAPHQL_WS_ROOT_PATH, GraphQLSubscripti
on::new(schema.clone()))
        .layer(
```

```
            JwtAuthorizer::<RegisteredClaims>::from_secret(con
figuration.token.signing_secret.expose_secret().as_str())
                .build()
                .await
                .wrap_err("failed to create jwt_authorizer lay
er")?
                .into_layer()
                .allow_missing_token(true),
        )
        .layer(Extension(schema))
        .layer(
            CorsLayer::new()
                .allow_origin(configuration.rest.allowed_origi
n.parse::<HeaderValue>()?)
                .allow_headers(Any)
                .allow_methods(Any),
        )
        .route(HEALTH_PATH, routing::get(health::check)))
}
```

SQL

sqlx

Example of a function in `db.rs` of `accounts` service:k

```
pub async fn accounts_insert(
    user_id: Uuid,
    address: &AccountAddress,
    public_key: Option<String>,
    pool: &PgPool,
) -> Result<Account> {
    Ok(sqlx::query_as!(
        Account,
```

```
        r#"INSERT INTO accounts.accounts (chain, on_chain_addr
ess, public_key, user_id)
            VALUES ($1::accounts.chain, $2, $3, $4)
            ON CONFLICT (chain, on_chain_address) WHERE verifi
ed DO UPDATE
            SET user_id = $4
            RETURNING id, chain as "chain: Chain", on_chain_ad
dress, public_key, user_id"#,
        address.chain() as Chain,
        address.on_chain_address(),
        public_key,
        user_id,
    )
    .fetch_one(pool)
    .await?)
}
```

How to make type usable in database layer:

```
/// Ethereum, Aptos and Polygon use 256-bit addresses so for n
ow it is just a wrapper on top of it.
#[derive(
    Debug, Clone, UpperHex, LowerHex, Eq, PartialEq, PartialOr
d, Ord, Hash, SerializeDisplay, DeserializeFromStr,
)]
pub enum Address {
    EthAddress(EvmAddress),
    PolygonAddress(EvmAddress),
    AptosAddress(AptosAddress),
}
...
```

```
impl<'r> Decode<'r, Postgres> for Address {
    fn decode(value: PgValueRef) -> Result<Self, sqlx::error::
BoxDynError> {
        Ok(Self::from_str(value.as_str()?)?)
    }
}

// TODO: put under sqlx cfg feature
impl Type<Postgres> for Address {
    fn type_info() -> PgTypeInfo {
        PgTypeInfo::with_name("TEXT")
    }

    fn compatible(ty: &PgTypeInfo) -> bool {
        [<String as Type<Postgres>>::type_info()].contains(ty)
    }
}
```

Error Handling

eyre

```
// New Error
_ => Err(eyre!("wrong address format: {}", str_address)),

// Option to Error
.ok_or_else(|| eyre!("incorrect format: {}", str_address))?;

// Error propagation "as is"
Ok(Self::EthAddress(H160::from_str(eth_address.trim_start_matc
hes("0x"))?))

// Error wrapping with additional information
let id = Some(
```

```
    user_id_string
        .parse()
        .wrap_err("failed to parse user id from refresh token
subject")?,
);
```

Tracing

tracing

```
#[instrument(err(Debug), skip(pool), ret, level = "trace")]
pub async fn accounts_remove(user_id: Uuid, address: &Address,
pool: &PgPool) -> Result<bool> {
    db::accounts_delete(user_id, &address.on_chain_address(),
pool).await
}
...

#[instrument(err(Debug), skip(pool), ret, level = "trace")]
pub async fn accounts_delete(user_id: Uuid, on_chain_address:
&str, pool: &PgPool) -> Result<bool> {
    Ok(sqlx::query!(
        "DELETE FROM accounts.accounts WHERE user_id = $1 AND
on_chain_address = $2",
        user_id,
        on_chain_address
    )
    .execute(pool)
    .await?
    .rows_affected() ==
        1)
}
```

API

async-graphql

```
#[allow(clippy::missing_errors_doc)]
pub async fn schema() -> Result<Schema<Query, Mutation, EmptySubscription>> {
    let configuration = Configuration::from_env()?;
    let database_pool = database::connect(&configuration.database).await?;
    let client = Client::new();
    let daas_configuration = DaasConfiguration {
        base_path: DAAS_BASE_PATH.to_string(),
        client: client.clone(),
        ..Default::default()
    };
    Ok(Schema::build(Query, Mutation, EmptySubscription)
        .enable_federation()
        .enable_subscription_in_federation()
        .data(database_pool)
        .data(client)
        .data(configuration)
        .data(daas_configuration)
        .finish())
}
...
/// Add new unverified account to a list of accounts of the current user.
///
/// # Errors
///
/// This function will return an error if system error occurs.
#[instrument(err(Debug), skip(self, context), ret, level = "trace")]
```

```
async fn account_add<'a>(
    &self,
    context: &'a Context<'_>,
    account_address: Address,
    account_public_key: Option<String>,
) -> Result<AccountAddResponse> {
    let user_id = data::extract_user_id(context)?;
    let (configuration, pool) = extract_configuration_and_pool
_from_context(context)?;
    api::accounts_add(user_id, &account_address, account_publi
c_key, configuration, pool).await
}
```

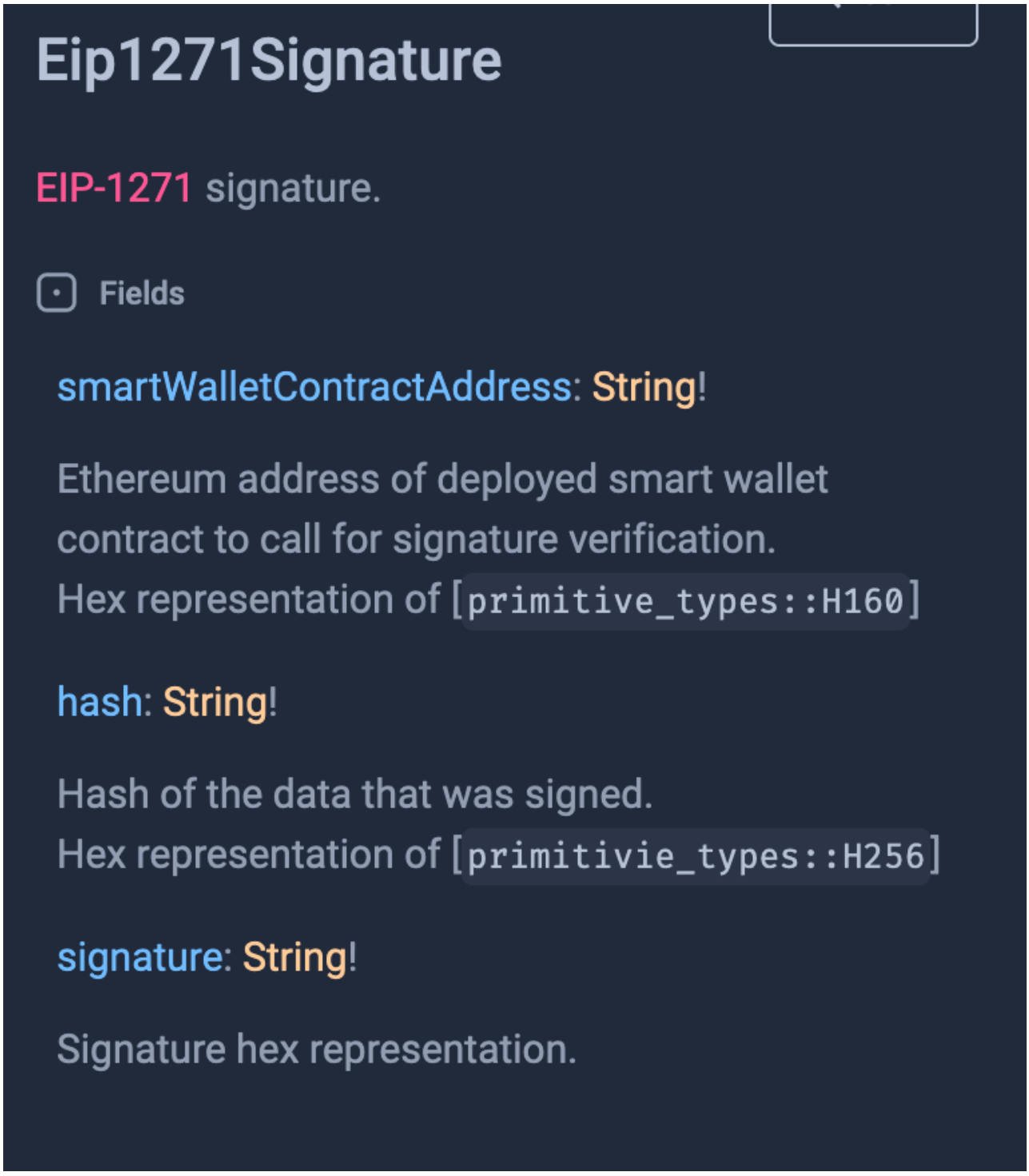
Usage of “scalar” types in API:

```
/// Ethereum, Aptos and Polygon use 256-bit addresses so for n
ow it is just a wrapper on top of it.
#[derive(
    Debug, Clone, UpperHex, LowerHex, Eq, PartialEq, PartialOr
d, Ord, Hash, SerializeDisplay, DeserializeFromStr,
)]
pub enum Address {
    EthAddress(EvmAddress),
    PolygonAddress(EvmAddress),
    AptosAddress(AptosAddress),
}
...
scalar!(Address, "AccountAddress");
```

Usage of complex types in API:

```
/// Different types of supported signatures.
```

```
#[derive(Clone, Debug, PartialEq, Eq, Serialize, Deserialize, OneofObject)]
#[serde(untagged)]
pub enum Signature {
    /// [EIP-1271](https://eips.ethereum.org/EIPS/eip-1271) signature.
    Eip1271(Eip1271Signature),
    /// [EIP-712](https://eips.ethereum.org/EIPS/eip-712) signature.
    Eip712(Eip712Signature),
    /// [Petra](https://petra.app/docs/signing-a-message) signature.
    Petra(PetraSignature),
}
...
/// [EIP-1271](https://eips.ethereum.org/EIPS/eip-1271) signature.
#[derive(Clone, Debug, PartialEq, Eq, Serialize, Deserialize, InputObject)]
pub struct Eip1271Signature {
    /// Ethereum address of deployed smart wallet contract to call for signature verification.
    /// Hex representation of [`primitive_types::H160`]
    smart_wallet_contract_address: String,
    /// Hash of the data that was signed.
    /// Hex representation of [`primitivie_types::H256`]
    hash: String,
    /// Signature hex representation.
    signature: String,
}
```



Time

time

```
let active2fa = if scopes.contains(&Scopes::TwoFactorAuth) {
    OffsetDateTime::now_utc()
        .saturating_add(self.active_2fa_time)
```

```
        .unix_timestamp()
} else {
    0
};
...
#[derive(Debug, sqlx::FromRow)]
pub struct OrderEvent {
    pub id: i64,
    pub order_id: Uuid,
    pub order_status: Option<OrderStatus>,
    pub payload: Option<String>,
    pub created_at: OffsetDateTime,
}
```

Utilities

structstruck

```
structstruck::strike! {
    #[strikethrough[derive(Debug, Clone, SimpleObject)]]
    pub struct Accounts {
    //! Details of user's accounts.
    //! Contains information about active accounts and their f
unds.
        /// List of active connected accounts.
        pub accounts: Vec<
            pub struct Account {
                pub address: Address,
                pub balance: pub struct Balance {
                    /// List of coins that belongs to the [`Ac
count`].
                    pub coins: HashMap<Currency, Coin>,
```

```
                    /// List of collections that belongs to th
e [`Account`].
                    pub collections: HashMap<CollectionId, Col
lection>,
                    /// List of collectibles that belongs to t
he [`Account`].
                    pub collectibles: HashMap<CollectibleId, C
ollectible>,
                },
            }
        >,
        pub cursor: Option<String>,
    }
}
```

serde_with

```
use serde_with::{DeserializeFromStr, SerializeDisplay};
...
/// Ethereum, Aptos and Polygon use 256-bit addresses so for n
ow it is just a wrapper on top of it.
#[derive(
    Debug, Clone, UpperHex, LowerHex, Eq, PartialEq, PartialOr
d, Ord, Hash, SerializeDisplay, DeserializeFromStr,
)]
pub enum Address {
    EthAddress(EvmAddress),
    PolygonAddress(EvmAddress),
    AptosAddress(AptosAddress),
}

impl Display for Address {
```

```
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fm
t::Result {
        write!(f, "{}.{}", self.chain(), self.on_chain_address
())
    }
}

impl FromStr for Address {
    type Err = ErrReport;

    fn from_str(str_address: &str) -> Result<Self, Self::Err>
{
        let (chain_str, address_str) = str_address
            .split_once('.')
            .ok_or_else(|| eyre!("incorrect format: {}", str_a
ddress))?;
        match (Chain::from_str(chain_str), address_str) {
            (Ok(Chain::Ethereum), eth_address) => {
                Ok(Self::EthAddress(H160::from_str(eth_addres
s.trim_start_matches("0x"))?))
            },
            (Ok(Chain::Polygon), polygon_address) => Ok(Self::
PolygonAddress(H160::from_str(
                polygon_address.trim_start_matches("0x"),
            )?)),
            (Ok(Chain::Aptos), aptos_address) => Ok(Self::Apto
sAddress(H256::from_str(
                aptos_address.trim_start_matches("0x"),
            )?)),
            _ => Err(eyre!("wrong address format: {}", str_add
ress)),
        }
```

```
    }
}
```

Simple scenario:

```
/// Unique Cross-chain identifier for a Collectible.
#[derive(Clone, Debug, SerializeDisplay, DeserializeFromStr, D
isplay, Eq, PartialEq, PartialOrd, Ord, Hash)]
pub enum Id {
    #[display("{}.{}.{}", chain, contract_address, token_id)]
    Evm {
        chain: Chain,
        contract_address: String,
        token_id: String,
    },
    #[display("aptos.{}.{}.{}", creator_address, collection_na
me, token_name)]
    Aptos {
        creator_address: String,
        collection_name: String,
        token_name: String,
    },
}
```

Lints

```
#![deny(
    explicit_outlives_requirements,
    macro_use_extern_crate,
    missing_debug_implementations,
    trivial_casts,
    trivial_numeric_casts,
    unreachable_pub,
```

```
    unsafe_code,
    unused_qualifications,
    unused_results,
    variant_size_differences,
    unused_variables,
    clippy::complexity,
    clippy::nursery,
    clippy::pedantic,
    clippy::perf,
    clippy::style,
    clippy::suspicious,
    clippy::clone_on_ref_ptr,
    clippy::create_dir,
    clippy::dbg_macro,
    clippy::default_numeric_fallback,
    clippy::else_if_without_else,
    clippy::empty_structs_with_brackets,
    clippy::expect_used,
    clippy::get_unwrap,
    clippy::let_underscore_must_use,
    clippy::map_err_ignore,
    clippy::multiple_inherent_impl,
    clippy::panic,
    clippy::panic_in_result_fn,
    clippy::pub_use,
    clippy::rc_mutex,
    clippy::rest_pat_in_fully_bound_structs,
    clippy::same_name_method,
    clippy::self_named_module_files,
    clippy::shadow_reuse,
```

```
    clippy::shadow_same,
    clippy::shadow_unrelated,
    clippy::unseparated_literal_suffix,
    clippy::string_to_string,
    clippy::todo,
    clippy::unimplemented,
    clippy::unreachable,
    clippy::unwrap_in_result,
    clippy::unwrap_used,
    clippy::use_debug,
    clippy::verbose_file_reads,
    clippy::wildcard_enum_match_arm
)]
```

Format

```
newline_style="Unix"
comment_width = 120
max_width = 120
binop_separator = "Back"
use_small_heuristics = "default"
format_strings = true
hard_tabs = false
imports_layout = "HorizontalVertical"
imports_granularity = "Crate"
match_block_trailing_comma = true
normalize_comments = true
reorder_imports = true
reorder_modules = true
reorder_impl_items = true
```

```
space_after_colon = true
space_before_colon = false
struct_lit_single_line = true
use_field_init_shorthand = true
use_try_shorthand = true
unstable_features = true
format_code_in_doc_comments = true
where_single_line = true
wrap_comments = true
overflow_delimited_expr = true
edition = "2021"
ignore=[]
```

Links & Resources

- https://survey.stackoverflow.co/2023/
- https://blog.rust-lang.org/2023/08/07/Rust-Survey-2023-Results.html
- https://blog.logrocket.com/top-rust-web-frameworks/
- https://www.arewewebyet.org/