Validator Architecture

March 24th

PRYSM

A Product of Offchain Labs

Quick Intro

X: @jameshe_eth

Github: @james-prysm

Linkedin: jameshe2

Prysm Team @ Offchain Labs

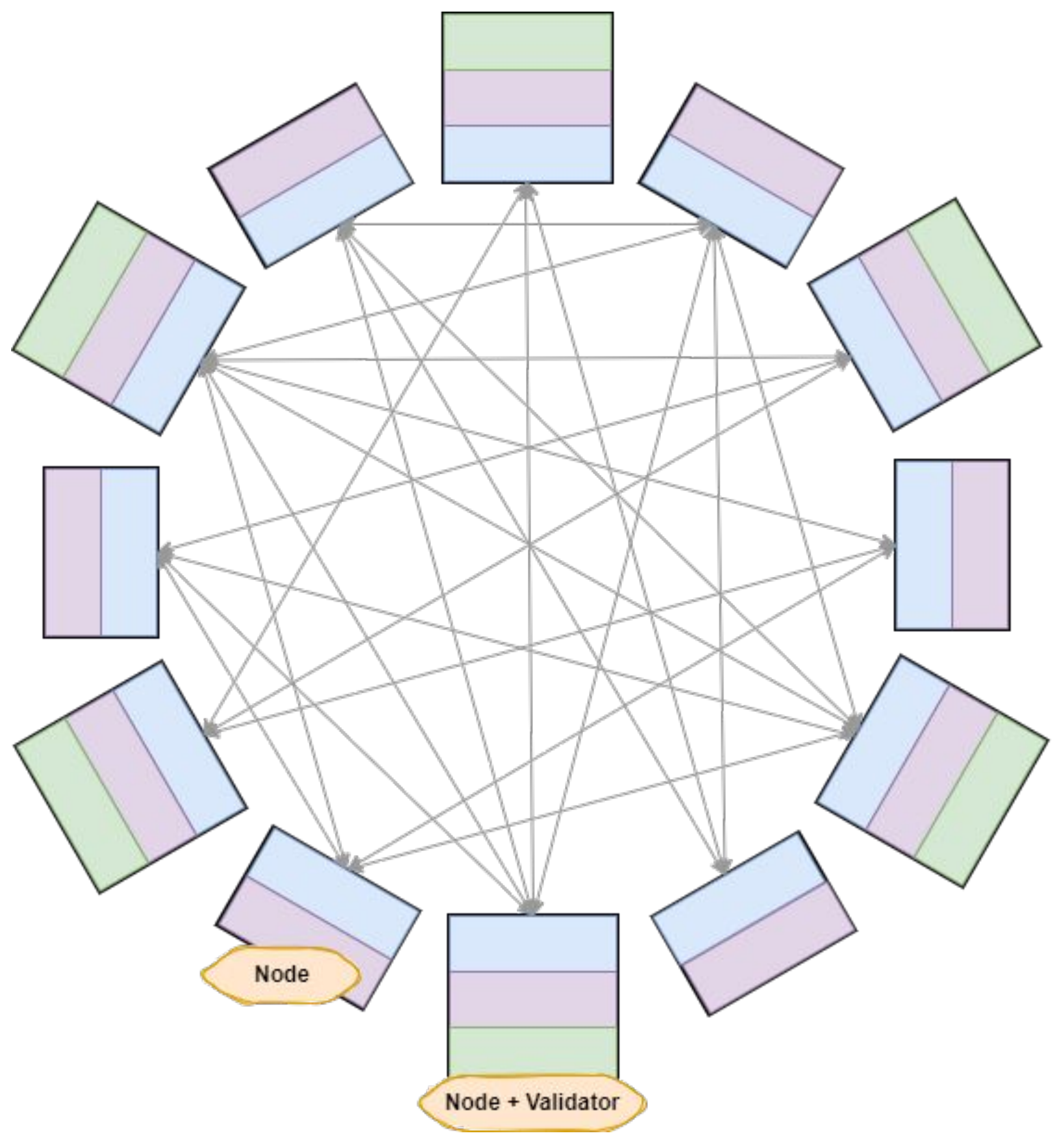
Prev: JPMC

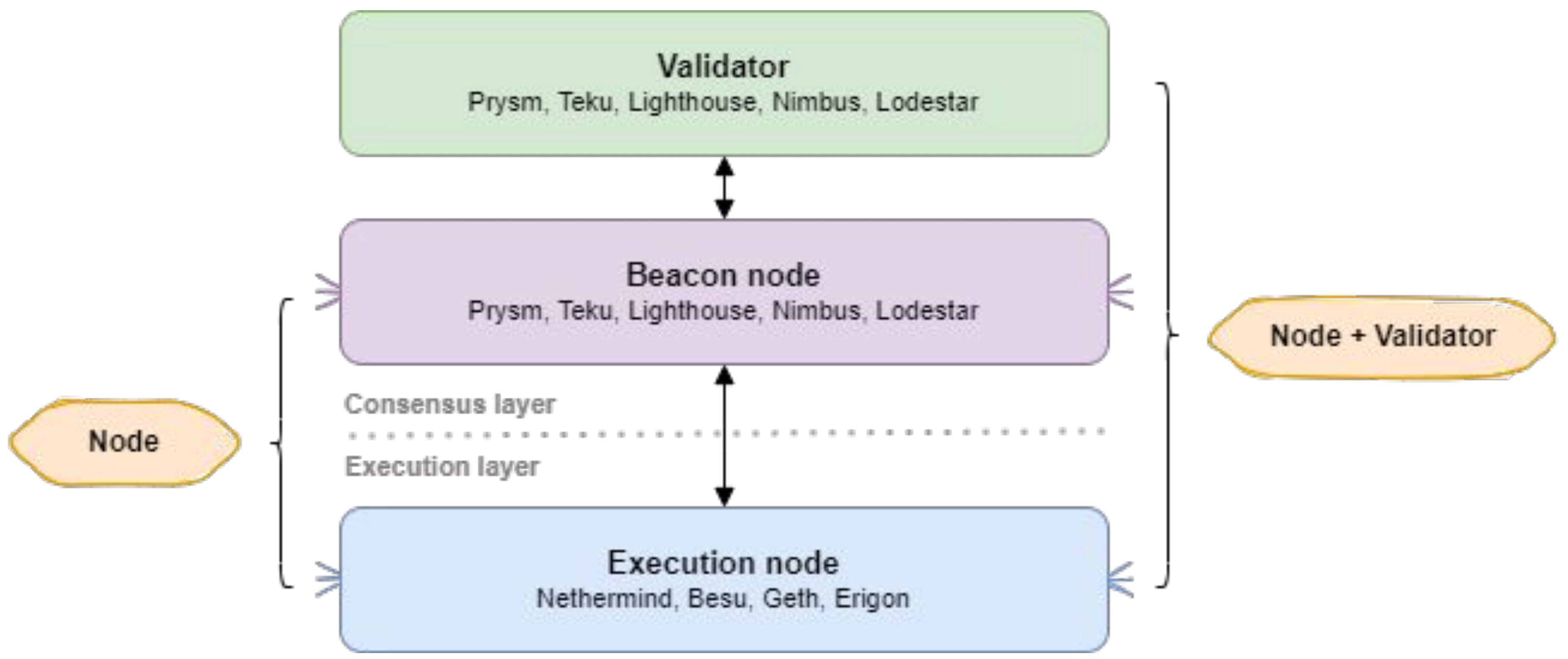
Refresher

PRYSM

A Product of Offchain Labs

Ethereum Node Architecture




Validator
Prysm, Teku, Lighthouse, Nimbus, Lodestar
Beacon node
Prysm, Teku, Lighthouse, Nimbus, Lodestar
Consensus layer
Execution layer
Execution node
Nethermind, Besu, Geth, Erigon
Node
Node + Validator

PRYSM
A Product of Offchain Labs

Purpose of the validator client

1. **Manage/Use** your validator **keystores**

- Import
- Backup
- Edit settings
- Sign
- Remove
- Slashing Protection
- ...

- JSON files that store your public and private validator key information
- Generated from the deposit CLI

2. Perform validator **protocol duties**

- Propose blocks
- Attest
- Aggregate
- Sync Committee

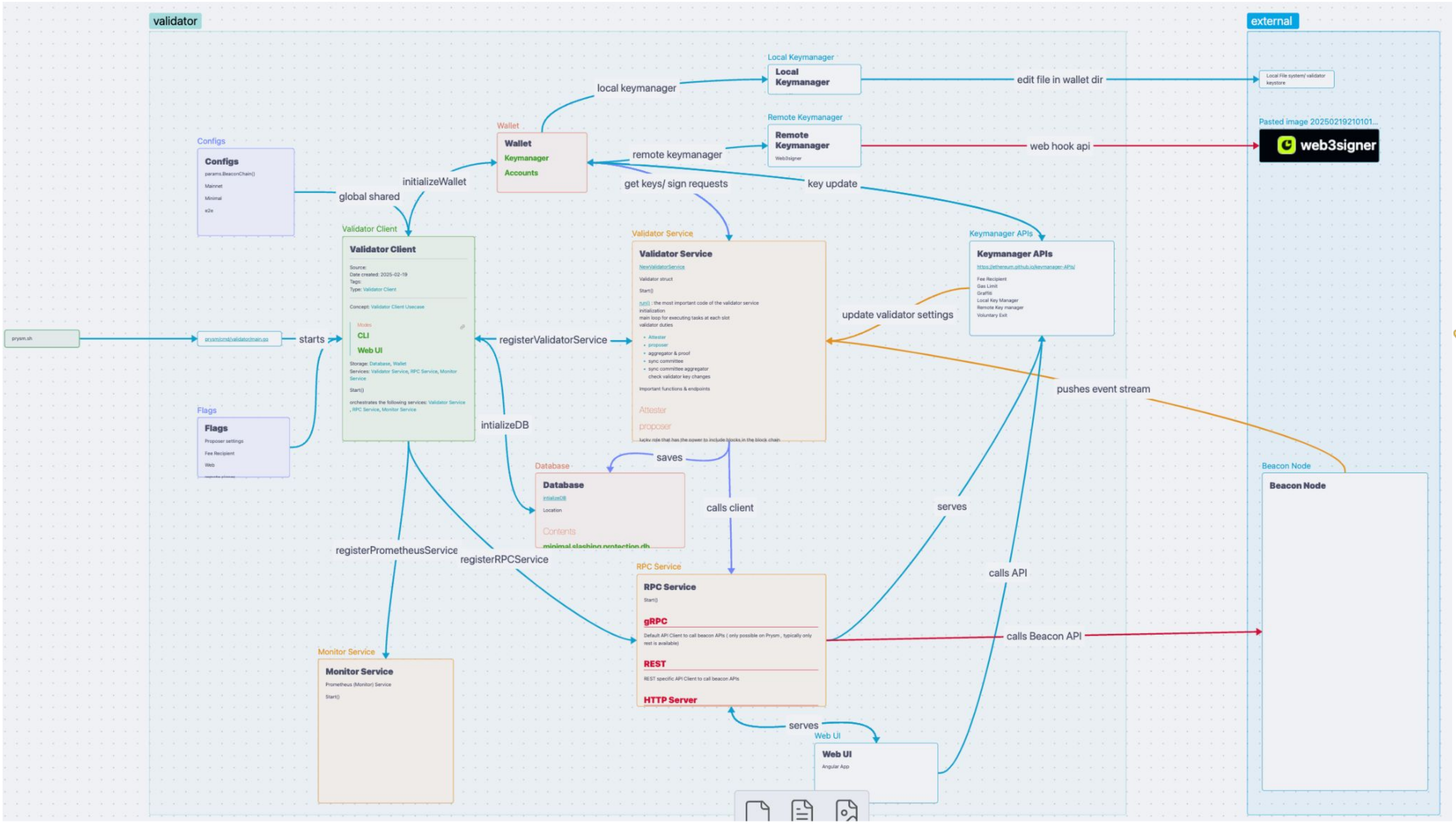
PRYSM
A Product of Offchain Labs

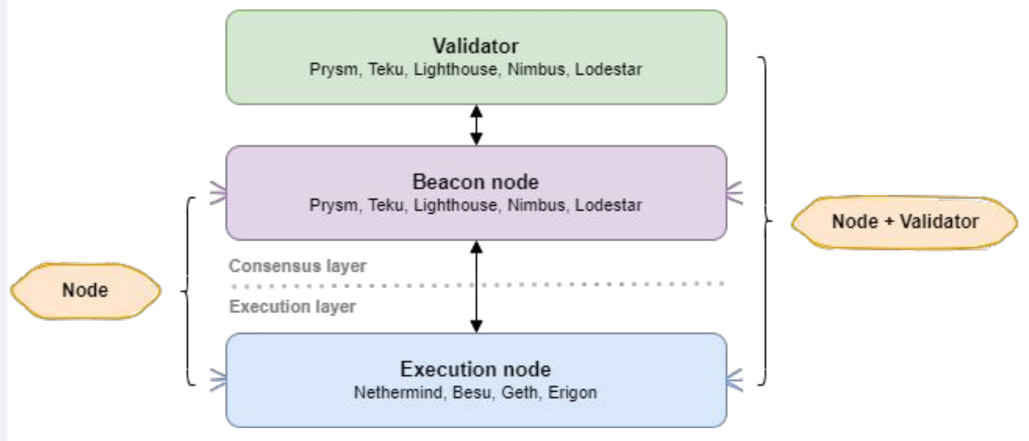
Prysm Validator Client

PRYSM

A Product of Offchain Labs

High level view

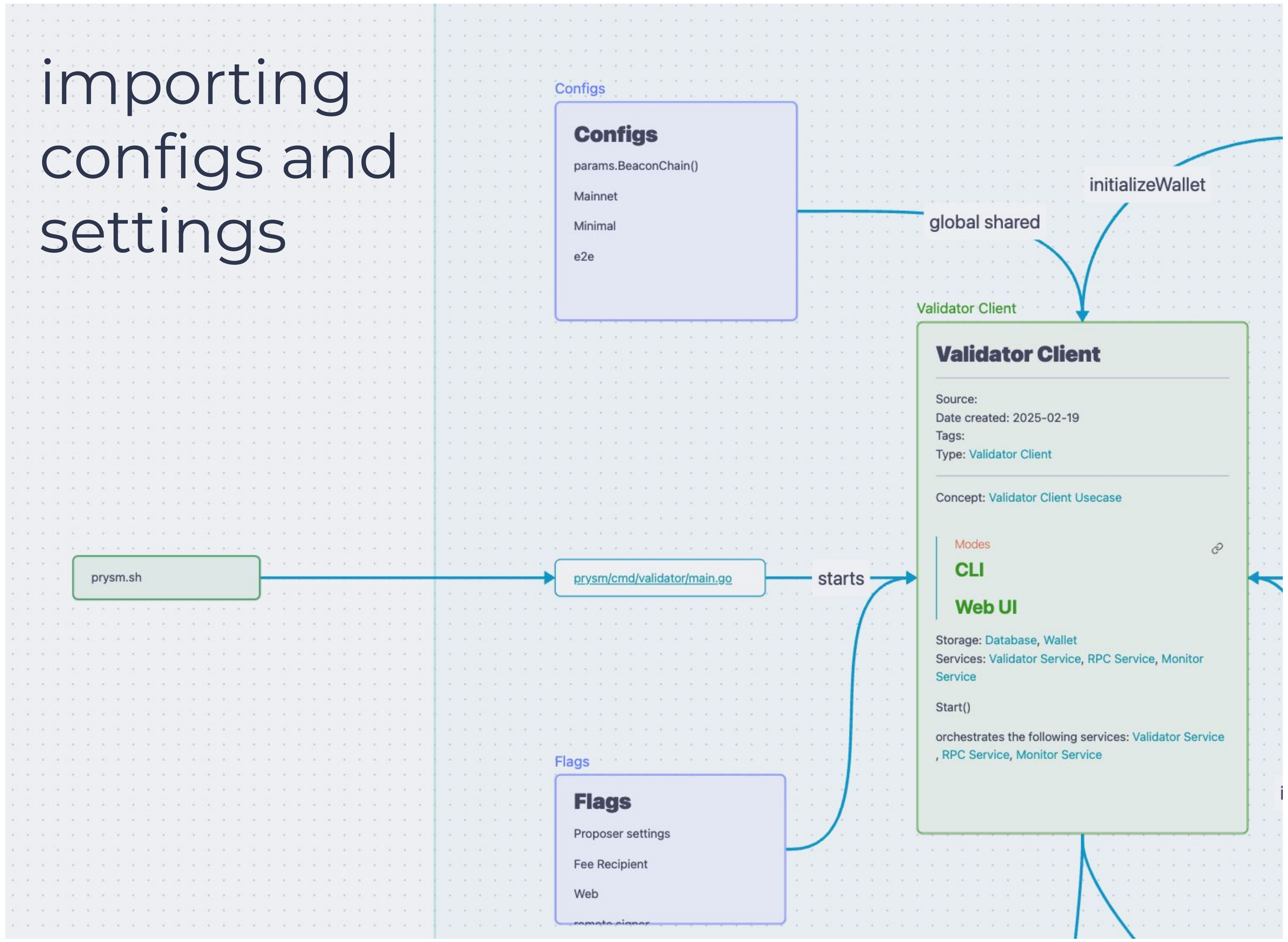


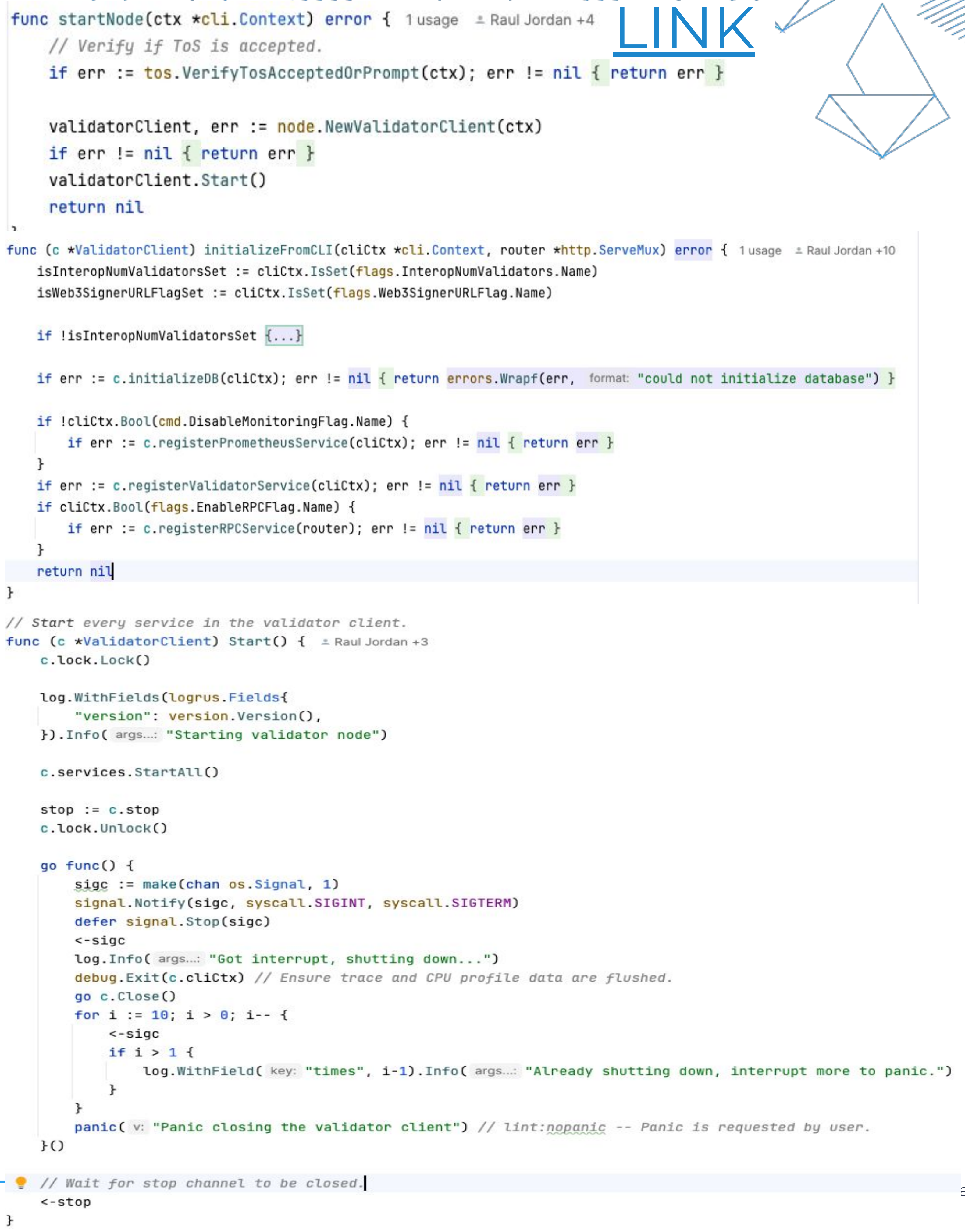

Validator
Prysm, Teku, Lighthouse, Nimbus, Lodestar
Beacon node
Prysm, Teku, Lighthouse, Nimbus, Lodestar
Consensus layer
Execution layer
Execution node
Nethermind, Besu, Geth, Erigon
Node
Node + Validator

PRYSM
A Product of Offchain Labs

LINK

Validator Client Type



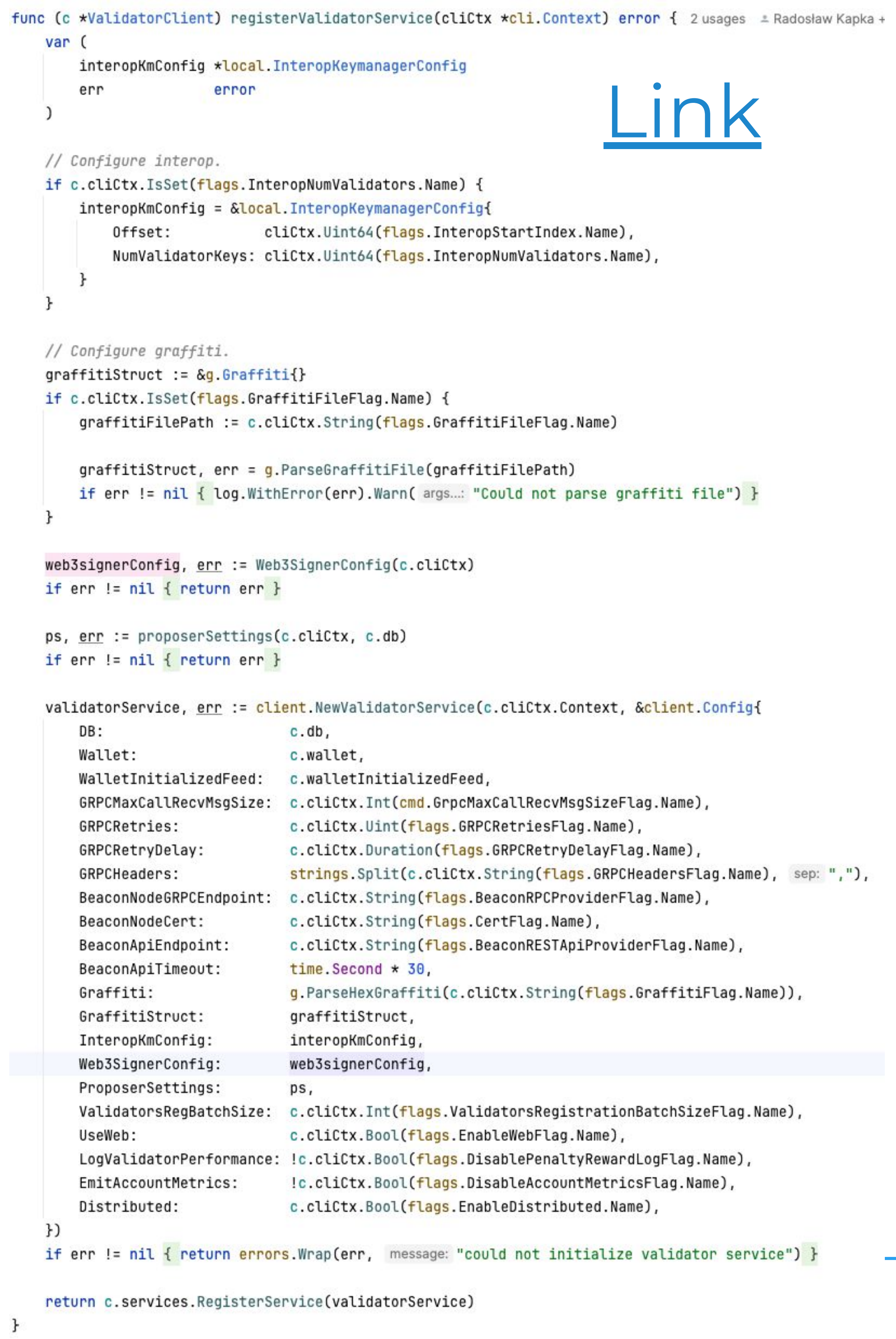


```go
func startNode(ctx *cli.Context) error {  1 usage  Raul Jordan +4
    // Verify if ToS is accepted.
    if err := tos.VerifyTosAcceptedOrPrompt(ctx); err != nil { return err }

    validatorClient, err := node.NewValidatorClient(ctx)
    if err != nil { return err }
    validatorClient.Start()
    return nil
}
```

```go
func (c *ValidatorClient) initializeFromCLI(cliCtx *cli.Context, router *http.ServeMux) error {  1 usage  Raul Jordan +10
    isInteropNumValidatorsSet := cliCtx.IsSet(flags.InteropNumValidators.Name)
    isWeb3SignerURLFlagSet := cliCtx.IsSet(flags.Web3SignerURLFlag.Name)

    if !isInteropNumValidatorsSet {...}

    if err := c.initializeDB(cliCtx); err != nil { return errors.Wrapf(err,  format: "could not initialize database") }

    if !cliCtx.Bool(cmd.DisableMonitoringFlag.Name) {
        if err := c.registerPrometheusService(cliCtx); err != nil { return err }
    }
    if err := c.registerValidatorService(cliCtx); err != nil { return err }
    if cliCtx.Bool(flags.EnableRPCFlag.Name) {
        if err := c.registerRPCService(router); err != nil { return err }
    }
    return nil
}
```

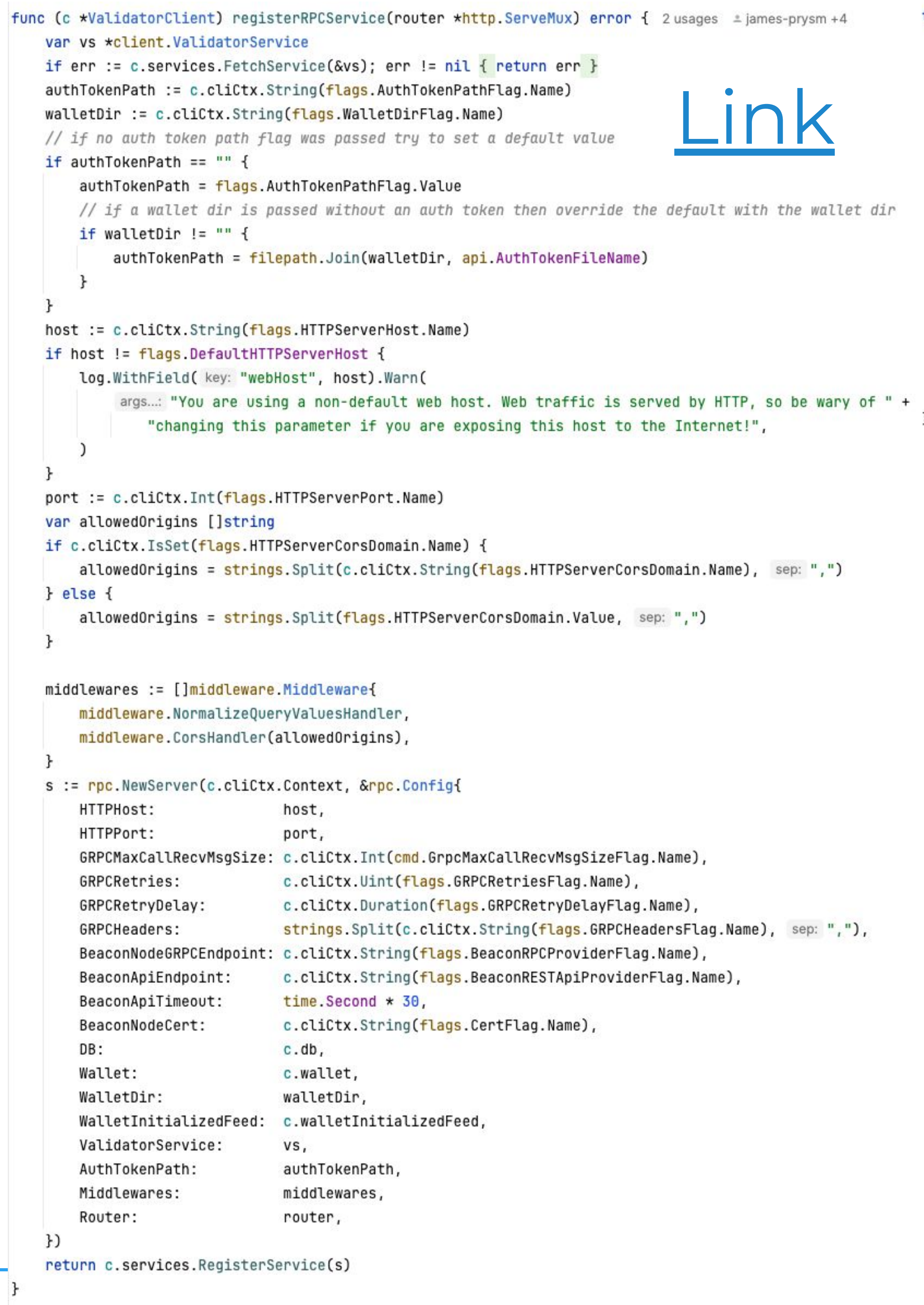
```go
// Start every service in the validator client.
func (c *ValidatorClient) Start() {  Raul Jordan +3
    c.lock.Lock()

    log.WithFields(logrus.Fields{
        "version": version.Version(),
    }).Info( args...: "Starting validator node")

    c.services.StartAll()

    stop := c.stop
    c.lock.Unlock()

    go func() {
        sigc := make(chan os.Signal, 1)
        signal.Notify(sigc, syscall.SIGINT, syscall.SIGTERM)
        defer signal.Stop(sigc)
        <-sigc
        log.Info( args...: "Got interrupt, shutting down...")
        debug.Exit(c.cliCtx) // Ensure trace and CPU profile data are flushed.
        go c.Close()
        for i := 10; i > 0; i-- {
            <-sigc
            if i > 1 {
                log.WithField( key: "times", i-1).Info( args...: "Already shutting down, interrupt more to panic.")
            }
        }
        panic( v: "Panic closing the validator client") // lint:nopanic -- Panic is requested by user.
    }()

    // Wait for stop channel to be closed.
    <-stop
}
```

Registered Services



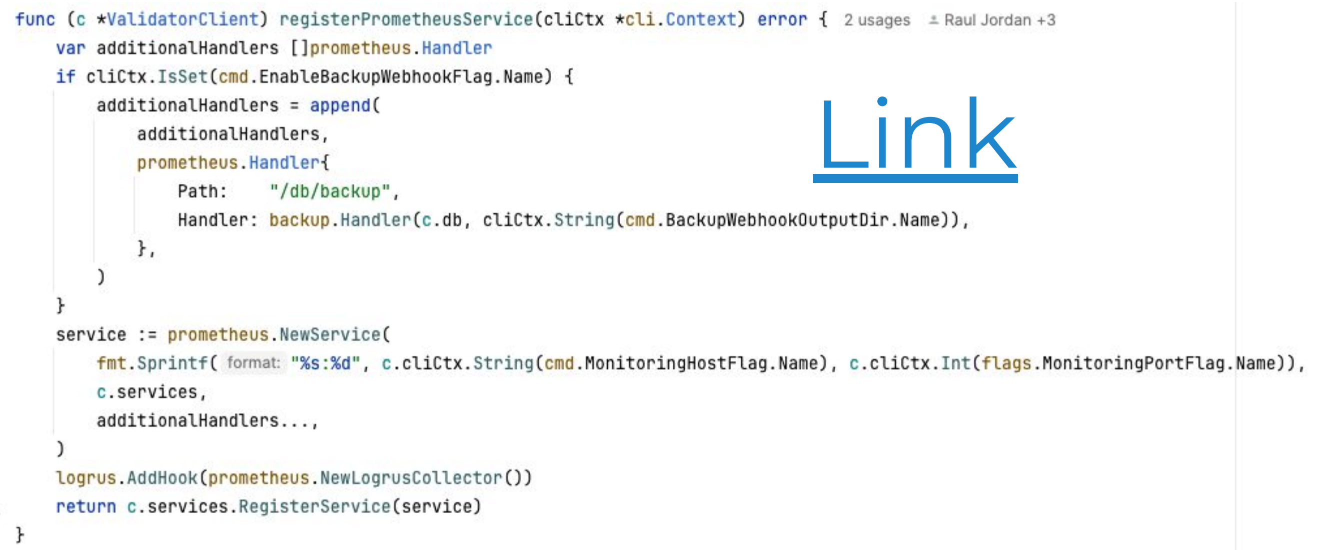
Link

```go
func (c *ValidatorClient) registerValidatorService(cliCtx *cli.Context) error {  2 usages  ≛ Radosław Kapka +
    var (
        interopKmConfig *local.InteropKeymanagerConfig
        err             error
    )

    // Configure interop.
    if c.cliCtx.IsSet(flags.InteropNumValidators.Name) {
        interopKmConfig = &local.InteropKeymanagerConfig{
            Offset:           cliCtx.Uint64(flags.InteropStartIndex.Name),
            NumValidatorKeys: cliCtx.Uint64(flags.InteropNumValidators.Name),
        }
    }

    // Configure graffiti.
    graffitiStruct := &g.Graffiti{}
    if c.cliCtx.IsSet(flags.GraffitiFileFlag.Name) {
        graffitiFilePath := c.cliCtx.String(flags.GraffitiFileFlag.Name)

        graffitiStruct, err = g.ParseGraffitiFile(graffitiFilePath)
        if err != nil { log.WithError(err).Warn( args...: "Could not parse graffiti file") }
    }

    web3signerConfig, err := Web3SignerConfig(c.cliCtx)
    if err != nil { return err }

    ps, err := proposerSettings(c.cliCtx, c.db)
    if err != nil { return err }

    validatorService, err := client.NewValidatorService(c.cliCtx.Context, &client.Config{
        DB:                      c.db,
        Wallet:                  c.wallet,
        WalletInitializedFeed:   c.walletInitializedFeed,
        GRPCMaxCallRecvMsgSize:  c.cliCtx.Int(cmd.GrpcMaxCallRecvMsgSizeFlag.Name),
        GRPCRetries:             c.cliCtx.Uint(flags.GRPCRetriesFlag.Name),
        GRPCRetryDelay:          c.cliCtx.Duration(flags.GRPCRetryDelayFlag.Name),
        GRPCHeaders:             strings.Split(c.cliCtx.String(flags.GRPCHeadersFlag.Name),  sep: ","),
        BeaconNodeGRPCEndpoint:  c.cliCtx.String(flags.BeaconRPCProviderFlag.Name),
        BeaconNodeCert:          c.cliCtx.String(flags.CertFlag.Name),
        BeaconApiEndpoint:       c.cliCtx.String(flags.BeaconRESTApiProviderFlag.Name),
        BeaconApiTimeout:        time.Second * 30,
        Graffiti:                g.ParseHexGraffiti(c.cliCtx.String(flags.GraffitiFlag.Name)),
        GraffitiStruct:          graffitiStruct,
        InteropKmConfig:         interopKmConfig,
        Web3SignerConfig:        web3signerConfig,
        ProposerSettings:        ps,
        ValidatorsRegBatchSize:  c.cliCtx.Int(flags.ValidatorsRegistrationBatchSizeFlag.Name),
        UseWeb:                  c.cliCtx.Bool(flags.EnableWebFlag.Name),
        LogValidatorPerformance: !c.cliCtx.Bool(flags.DisablePenaltyRewardLogFlag.Name),
        EmitAccountMetrics:      !c.cliCtx.Bool(flags.DisableAccountMetricsFlag.Name),
        Distributed:             c.cliCtx.Bool(flags.EnableDistributed.Name),
    })
    if err != nil { return errors.Wrap(err,  message: "could not initialize validator service") }

    return c.services.RegisterService(validatorService)
}
```



Link

```go
func (c *ValidatorClient) registerRPCService(router *http.ServeMux) error {  2 usages  ≛ james-prysm +4
    var vs *client.ValidatorService
    if err := c.services.FetchService(&vs); err != nil { return err }
    authTokenPath := c.cliCtx.String(flags.AuthTokenPathFlag.Name)
    walletDir := c.cliCtx.String(flags.WalletDirFlag.Name)
    // if no auth token path flag was passed try to set a default value
    if authTokenPath == "" {
        authTokenPath = flags.AuthTokenPathFlag.Value
        // if a wallet dir is passed without an auth token then override the default with the wallet dir
        if walletDir != "" {
            authTokenPath = filepath.Join(walletDir, api.AuthTokenFileName)
        }
    }
    host := c.cliCtx.String(flags.HTTPServerHost.Name)
    if host != flags.DefaultHTTPServerHost {
        log.WithField( key: "webHost", host).Warn(
             args...: "You are using a non-default web host. Web traffic is served by HTTP, so be wary of " +
                "changing this parameter if you are exposing this host to the Internet!",
        )
    }
    port := c.cliCtx.Int(flags.HTTPServerPort.Name)
    var allowedOrigins []string
    if c.cliCtx.IsSet(flags.HTTPServerCorsDomain.Name) {
        allowedOrigins = strings.Split(c.cliCtx.String(flags.HTTPServerCorsDomain.Name),  sep: ",")
    } else {
        allowedOrigins = strings.Split(flags.HTTPServerCorsDomain.Value,  sep: ",")
    }

    middlewares := []middleware.Middleware{
        middleware.NormalizeQueryValuesHandler,
        middleware.CorsHandler(allowedOrigins),
    }
    s := rpc.NewServer(c.cliCtx.Context, &rpc.Config{
        HTTPHost:               host,
        HTTPPort:               port,
        GRPCMaxCallRecvMsgSize: c.cliCtx.Int(cmd.GrpcMaxCallRecvMsgSizeFlag.Name),
        GRPCRetries:            c.cliCtx.Uint(flags.GRPCRetriesFlag.Name),
        GRPCRetryDelay:         c.cliCtx.Duration(flags.GRPCRetryDelayFlag.Name),
        GRPCHeaders:            strings.Split(c.cliCtx.String(flags.GRPCHeadersFlag.Name),  sep: ","),
        BeaconNodeGRPCEndpoint: c.cliCtx.String(flags.BeaconRPCProviderFlag.Name),
        BeaconApiEndpoint:      c.cliCtx.String(flags.BeaconRESTApiProviderFlag.Name),
        BeaconApiTimeout:       time.Second * 30,
        BeaconNodeCert:         c.cliCtx.String(flags.CertFlag.Name),
        DB:                     c.db,
        Wallet:                 c.wallet,
        WalletDir:              walletDir,
        WalletInitializedFeed:  c.walletInitializedFeed,
        ValidatorService:       vs,
        AuthTokenPath:          authTokenPath,
        Middlewares:            middlewares,
        Router:                 router,
    })
    return c.services.RegisterService(s)
}
```

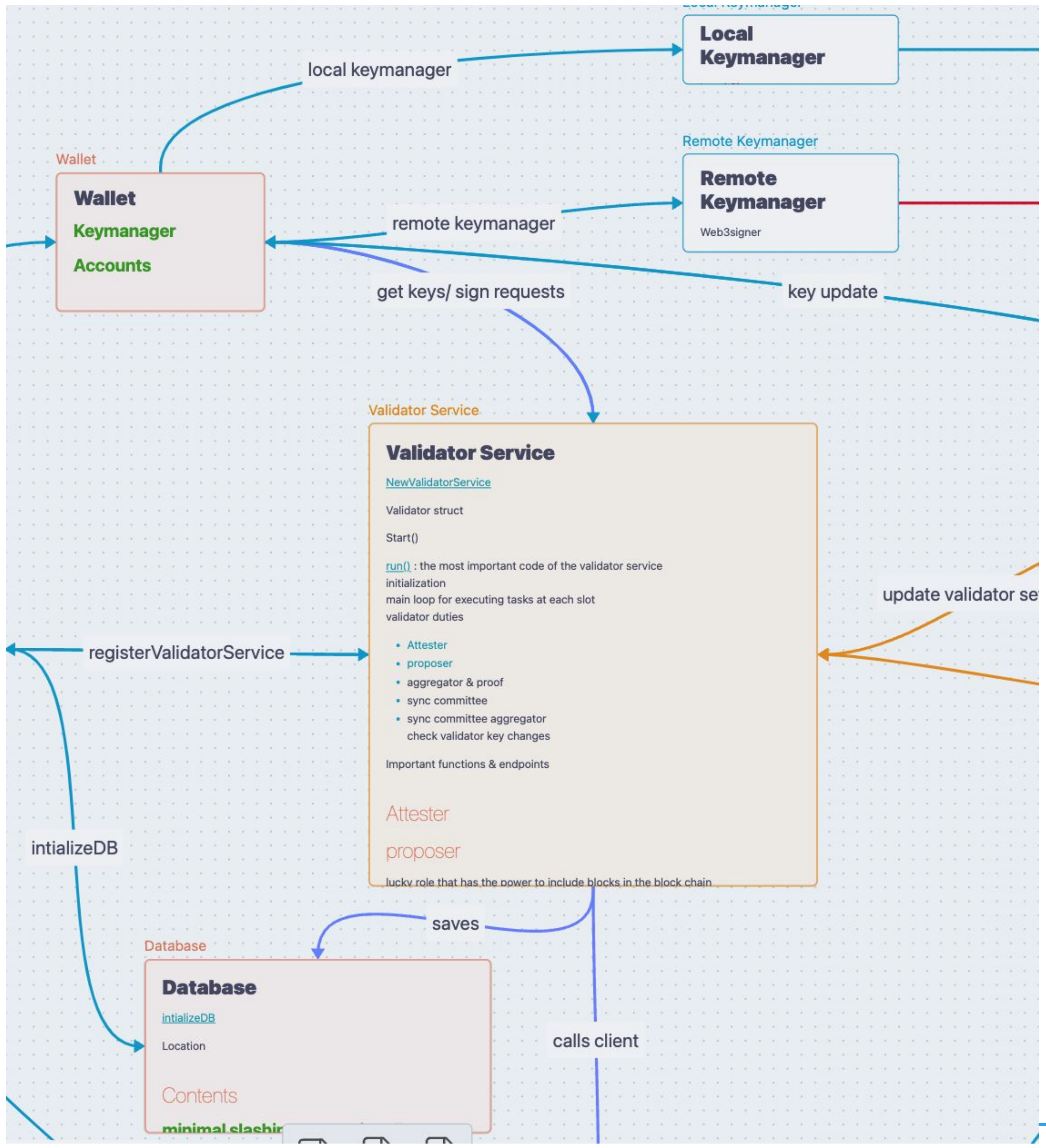


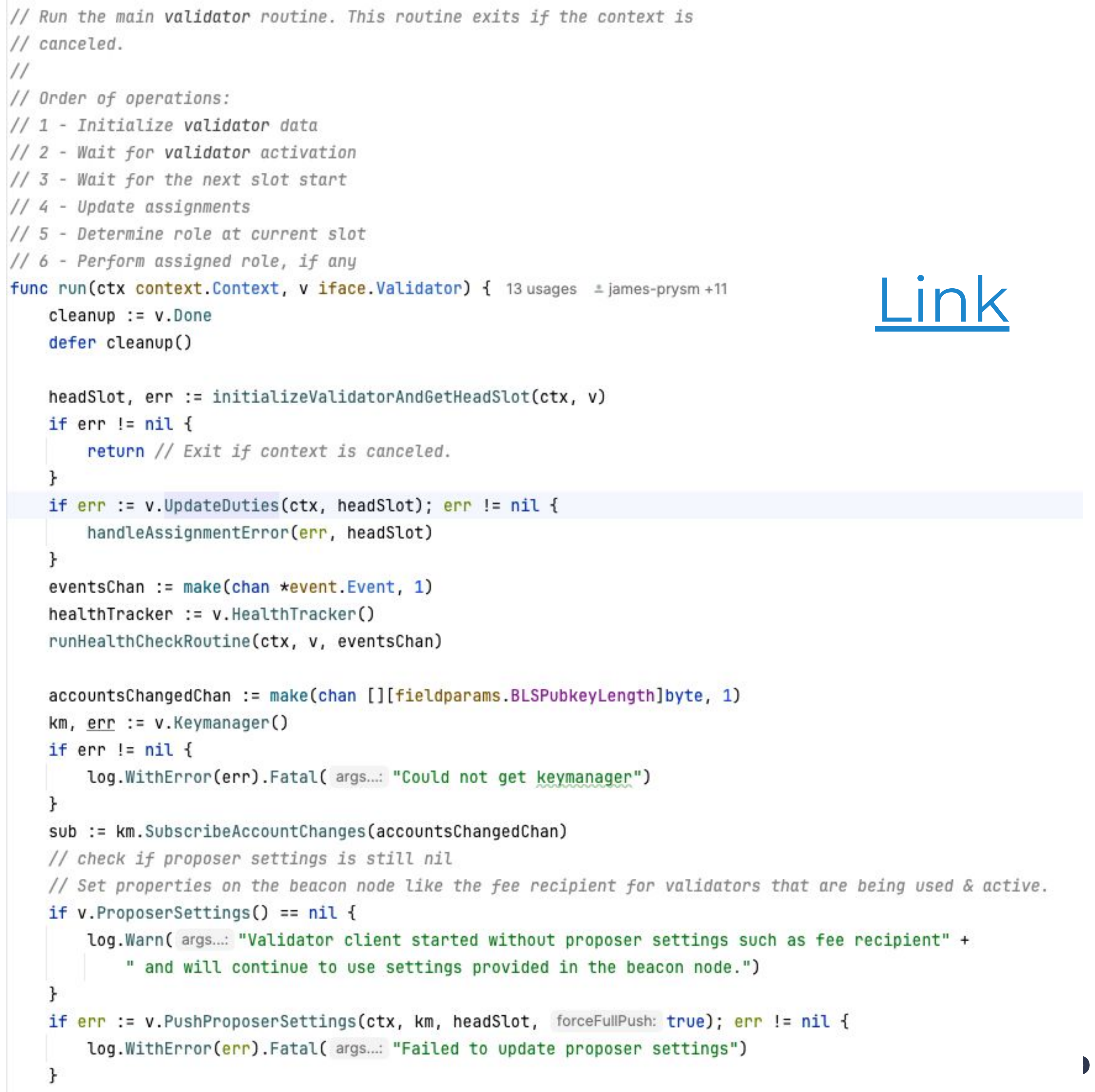
Link

```go
func (c *ValidatorClient) registerPrometheusService(cliCtx *cli.Context) error {  2 usages  ≛ Raul Jordan +3
    var additionalHandlers []prometheus.Handler
    if cliCtx.IsSet(cmd.EnableBackupWebhookFlag.Name) {
        additionalHandlers = append(
            additionalHandlers,
            prometheus.Handler{
                Path:    "/db/backup",
                Handler: backup.Handler(c.db, cliCtx.String(cmd.BackupWebhookOutputDir.Name)),
            },
        )
    }
    service := prometheus.NewService(
        fmt.Sprintf( format: "%s:%d", c.cliCtx.String(cmd.MonitoringHostFlag.Name), c.cliCtx.Int(flags.MonitoringPortFlag.Name)),
        c.services,
        additionalHandlers...,
    )
    logrus.AddHook(prometheus.NewLogrusCollector())
    return c.services.RegisterService(service)
}
```

PRYSM
A Product of Offchain Labs

Validator Service





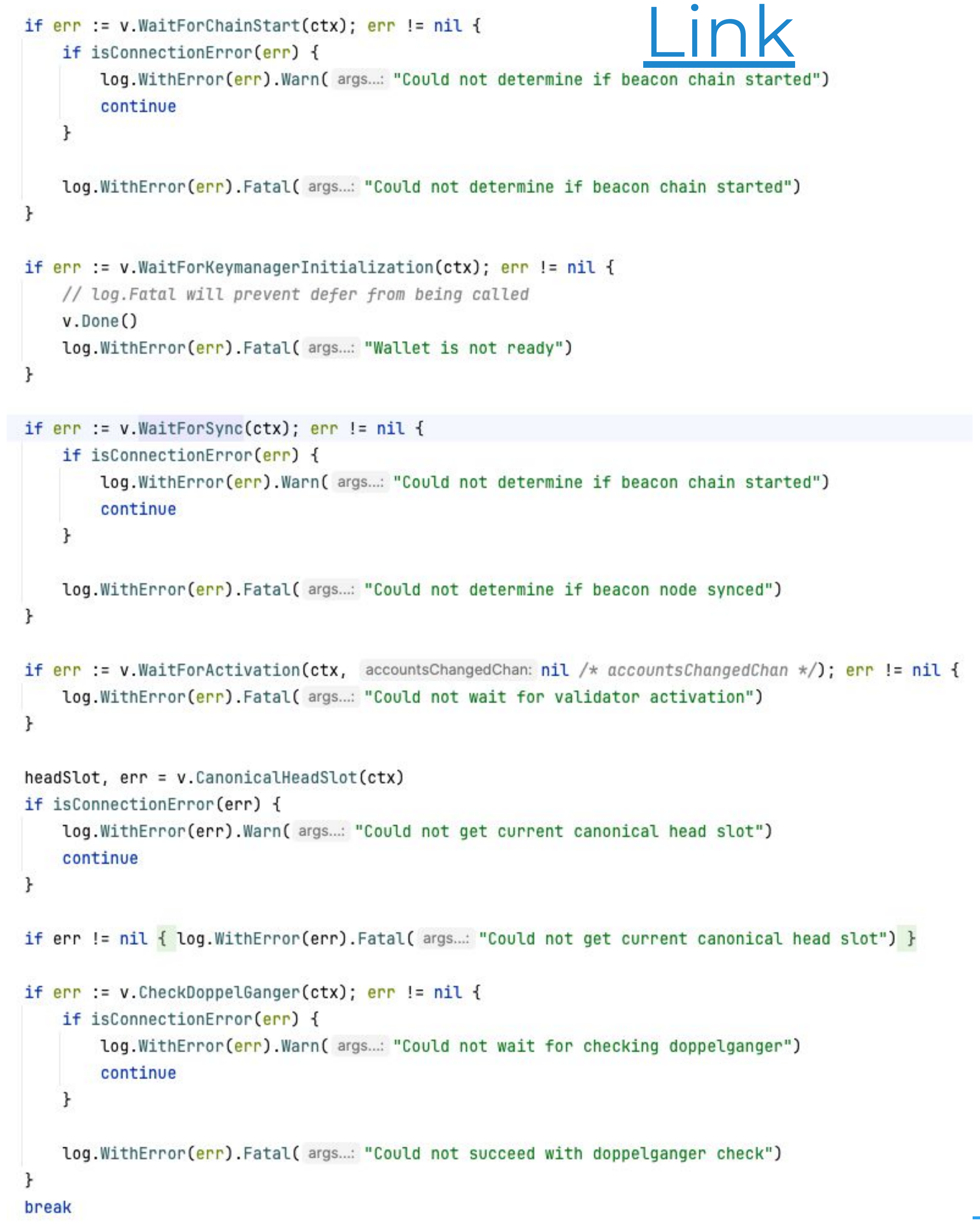
```go
// Run the main validator routine. This routine exits if the context is
// canceled.
//
// Order of operations:
// 1 - Initialize validator data
// 2 - Wait for validator activation
// 3 - Wait for the next slot start
// 4 - Update assignments
// 5 - Determine role at current slot
// 6 - Perform assigned role, if any
func run(ctx context.Context, v iface.Validator) {  13 usages  ± james-prysm +11
	cleanup := v.Done
	defer cleanup()

	headSlot, err := initializeValidatorAndGetHeadSlot(ctx, v)
	if err != nil {
		return // Exit if context is canceled.
	}
	if err := v.UpdateDuties(ctx, headSlot); err != nil {
		handleAssignmentError(err, headSlot)
	}
	eventsChan := make(chan *event.Event, 1)
	healthTracker := v.HealthTracker()
	runHealthCheckRoutine(ctx, v, eventsChan)

	accountsChangedChan := make(chan [][fieldparams.BLSPubkeyLength]byte, 1)
	km, err := v.Keymanager()
	if err != nil {
		log.WithError(err).Fatal( args...: "Could not get keymanager")
	}
	sub := km.SubscribeAccountChanges(accountsChangedChan)
	// check if proposer settings is still nil
	// Set properties on the beacon node like the fee recipient for validators that are being used & active.
	if v.ProposerSettings() == nil {
		log.Warn( args...: "Validator client started without proposer settings such as fee recipient" +
			" and will continue to use settings provided in the beacon node.")
	}
	if err := v.PushProposerSettings(ctx, km, headSlot,  forceFullPush: true); err != nil {
		log.WithError(err).Fatal( args...: "Failed to update proposer settings")
	}
```

Link

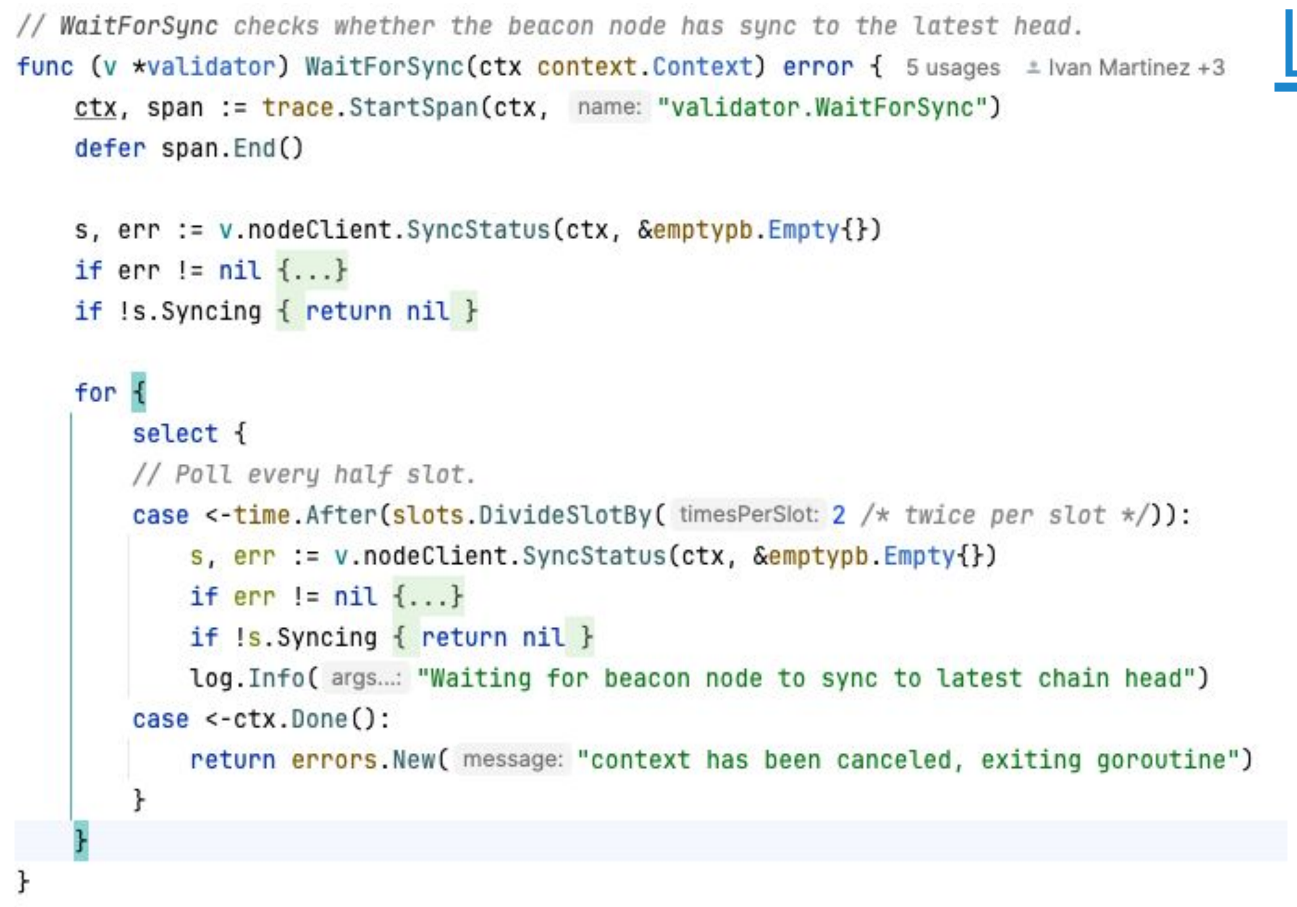
Validator Service Initialization

Link

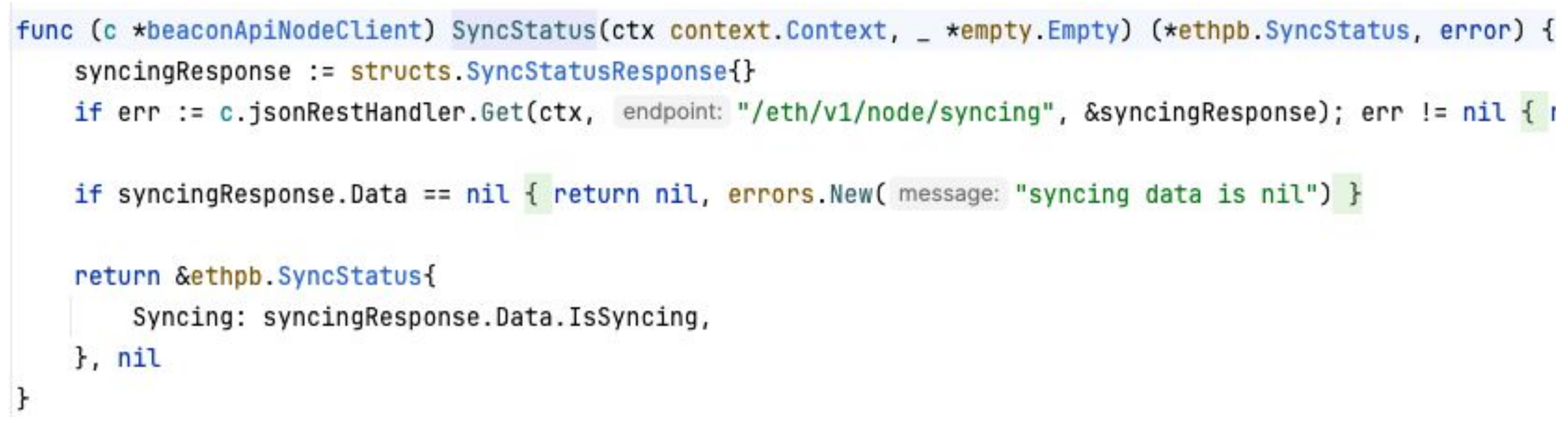
Link



```
if err := v.WaitForChainStart(ctx); err != nil {
    if isConnectionError(err) {
        log.WithError(err).Warn( args...: "Could not determine if beacon chain started")
        continue
    }

    log.WithError(err).Fatal( args...: "Could not determine if beacon chain started")
}

if err := v.WaitForKeymanagerInitialization(ctx); err != nil {
    // log.Fatal will prevent defer from being called
    v.Done()
    log.WithError(err).Fatal( args...: "Wallet is not ready")
}

if err := v.WaitForSync(ctx); err != nil {
    if isConnectionError(err) {
        log.WithError(err).Warn( args...: "Could not determine if beacon chain started")
        continue
    }

    log.WithError(err).Fatal( args...: "Could not determine if beacon node synced")
}

if err := v.WaitForActivation(ctx,  accountsChangedChan: nil /* accountsChangedChan */); err != nil {
    log.WithError(err).Fatal( args...: "Could not wait for validator activation")
}

headSlot, err = v.CanonicalHeadSlot(ctx)
if isConnectionError(err) {
    log.WithError(err).Warn( args...: "Could not get current canonical head slot")
    continue
}

if err != nil { log.WithError(err).Fatal( args...: "Could not get current canonical head slot") }

if err := v.CheckDoppelGanger(ctx); err != nil {
    if isConnectionError(err) {
        log.WithError(err).Warn( args...: "Could not wait for checking doppelganger")
        continue
    }

    log.WithError(err).Fatal( args...: "Could not succeed with doppelganger check")
}
break
```



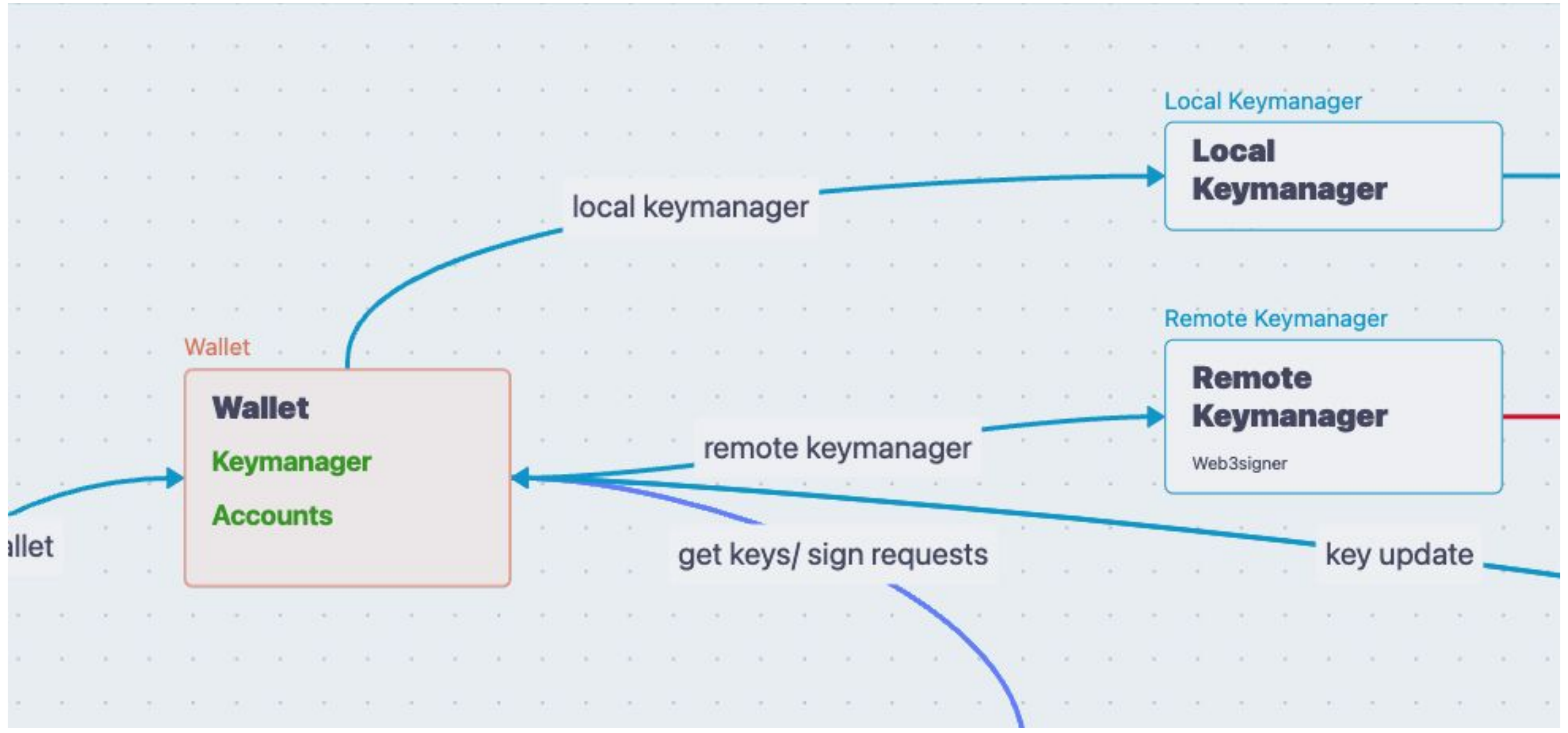
```
// WaitForSync checks whether the beacon node has sync to the latest head.
func (v *validator) WaitForSync(ctx context.Context) error {  5 usages  ± Ivan Martinez +3
    ctx, span := trace.StartSpan(ctx,  name: "validator.WaitForSync")
    defer span.End()

    s, err := v.nodeClient.SyncStatus(ctx, &emptypb.Empty{})
    if err != nil {...}
    if !s.Syncing { return nil }

    for {
        select {
        // Poll every half slot.
        case <-time.After(slots.DivideSlotBy( timesPerSlot: 2 /* twice per slot */)):
            s, err := v.nodeClient.SyncStatus(ctx, &emptypb.Empty{})
            if err != nil {...}
            if !s.Syncing { return nil }
            log.Info( args...: "Waiting for beacon node to sync to latest chain head")
        case <-ctx.Done():
            return errors.New( message: "context has been canceled, exiting goroutine")
        }
    }
}
```

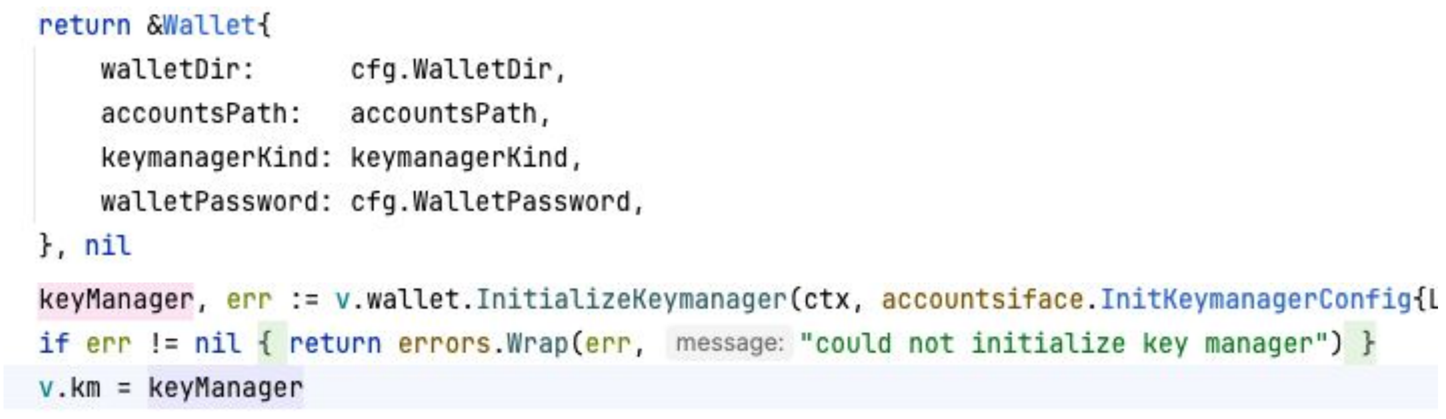


```
func (c *beaconApiNodeClient) SyncStatus(ctx context.Context, _ *empty.Empty) (*ethpb.SyncStatus, error) {
    syncingResponse := structs.SyncStatusResponse{}
    if err := c.jsonRestHandler.Get(ctx,  endpoint: "/eth/v1/node/syncing", &syncingResponse); err != nil {

    if syncingResponse.Data == nil { return nil, errors.New( message: "syncing data is nil") }

    return &ethpb.SyncStatus{
        Syncing: syncingResponse.Data.IsSyncing,
    }, nil
}
```

PRYSM
A Product of Offchain Labs

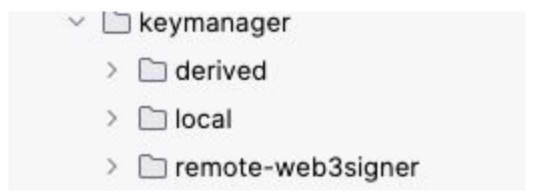
Wallet (local keymanager)





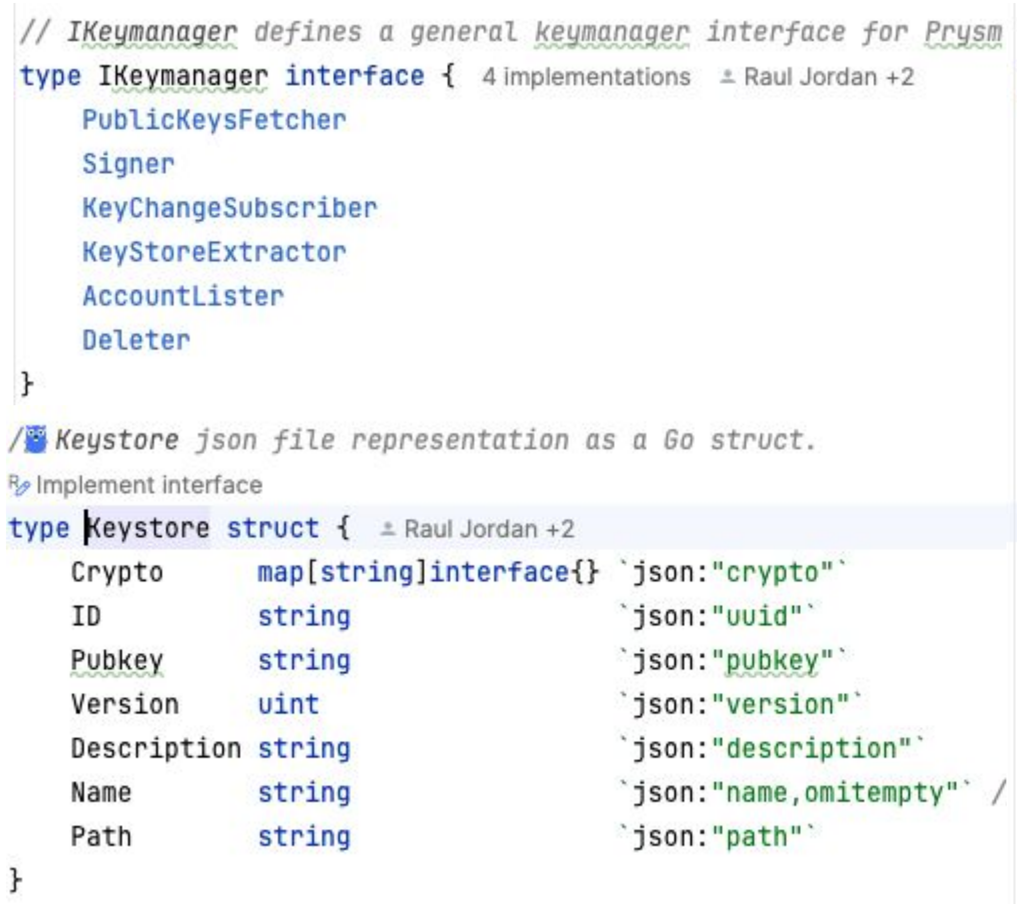
```go
return &Wallet{
    walletDir:      cfg.WalletDir,
    accountsPath:   accountsPath,
    keymanagerKind: keymanagerKind,
    walletPassword: cfg.WalletPassword,
}, nil
```

```go
keyManager, err := v.wallet.InitializeKeymanager(ctx, accountsiface.InitKeymanagerConfig{L
if err != nil { return errors.Wrap(err,  message: "could not initialize key manager") }
v.km = keyManager
```



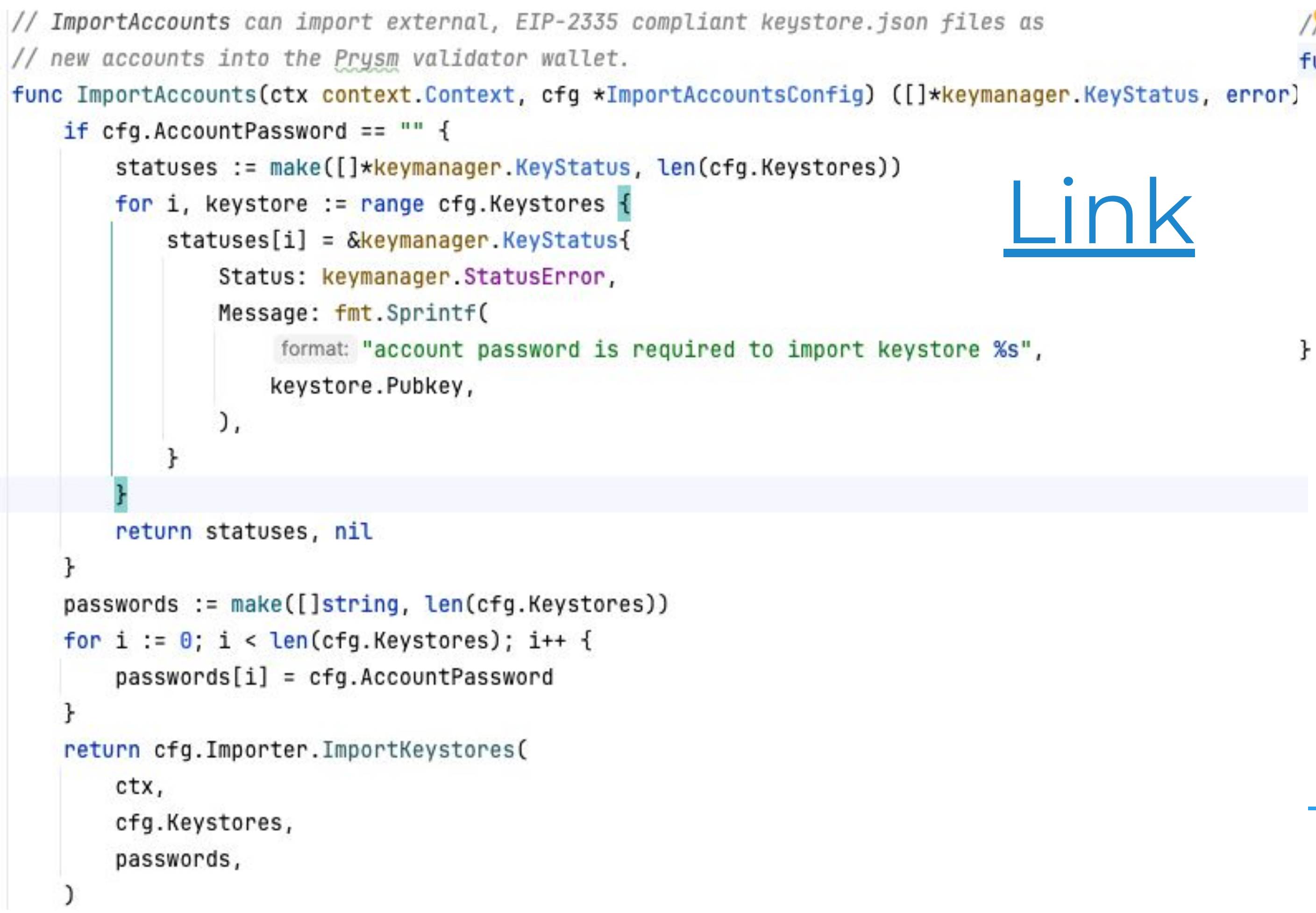
```
⌄ 🗀 keymanager
  › 🗀 derived
  › 🗀 local
  › 🗀 remote-web3signer
```

[Link]



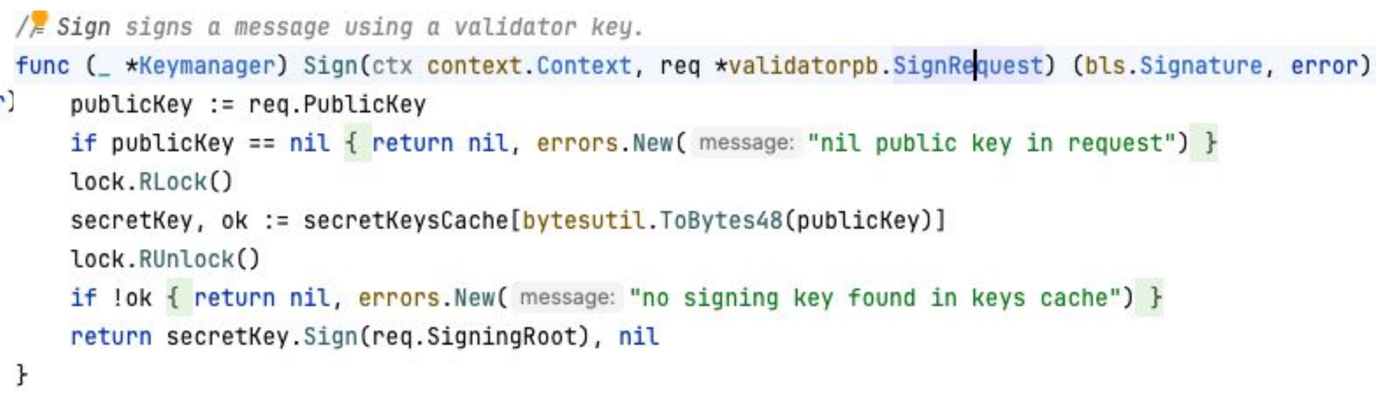
```go
// IKeymanager defines a general keymanager interface for Prysm
type IKeymanager interface {  4 implementations  ≗ Raul Jordan +2
    PublicKeysFetcher
    Signer
    KeyChangeSubscriber
    KeyStoreExtractor
    AccountLister
    Deleter
}
```

```go
// Keystore json file representation as a Go struct.
Implement interface
type Keystore struct {  ≗ Raul Jordan +2
    Crypto      map[string]interface{} `json:"crypto"`
    ID          string                 `json:"uuid"`
    Pubkey      string                 `json:"pubkey"`
    Version     uint                   `json:"version"`
    Description string                 `json:"description"`
    Name        string                 `json:"name,omitempty"` /
    Path        string                 `json:"path"`
}
```



```go
// ImportAccounts can import external, EIP-2335 compliant keystore.json files as
// new accounts into the Prysm validator wallet.
func ImportAccounts(ctx context.Context, cfg *ImportAccountsConfig) ([]*keymanager.KeyStatus, error)
    if cfg.AccountPassword == "" {
        statuses := make([]*keymanager.KeyStatus, len(cfg.Keystores))
        for i, keystore := range cfg.Keystores {
            statuses[i] = &keymanager.KeyStatus{
                Status: keymanager.StatusError,
                Message: fmt.Sprintf(
                     format: "account password is required to import keystore %s",
                    keystore.Pubkey,
                ),
            }
        }
        return statuses, nil
    }
    passwords := make([]string, len(cfg.Keystores))
    for i := 0; i < len(cfg.Keystores); i++ {
        passwords[i] = cfg.AccountPassword
    }
    return cfg.Importer.ImportKeystores(
        ctx,
        cfg.Keystores,
        passwords,
    )
```

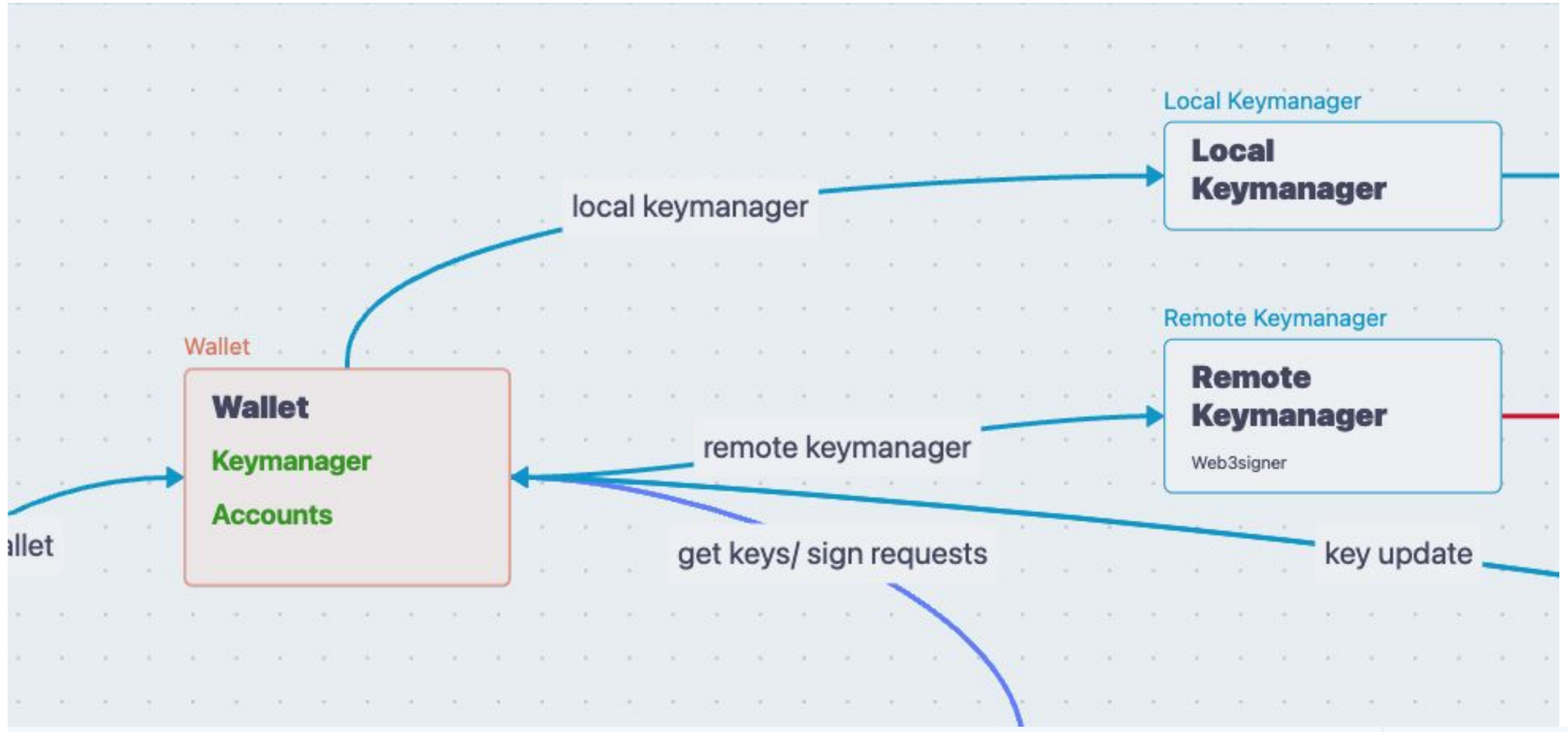
[Link]

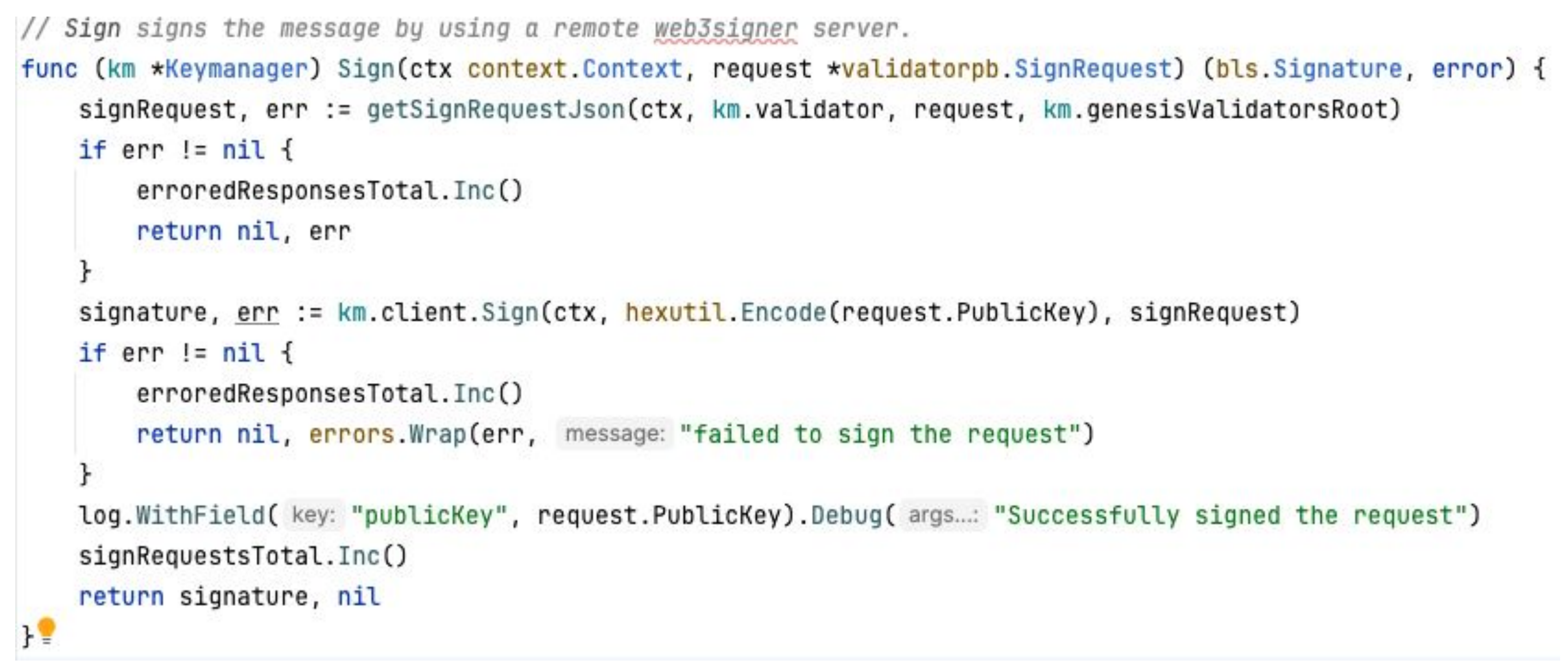


```go
// Sign signs a message using a validator key.
func (_ *Keymanager) Sign(ctx context.Context, req *validatorpb.SignRequest) (bls.Signature, error)
    publicKey := req.PublicKey
    if publicKey == nil { return nil, errors.New( message: "nil public key in request") }
    lock.RLock()
    secretKey, ok := secretKeysCache[bytesutil.ToBytes48(publicKey)]
    lock.RUnlock()
    if !ok { return nil, errors.New( message: "no signing key found in keys cache") }
    return secretKey.Sign(req.SigningRoot), nil
}
```

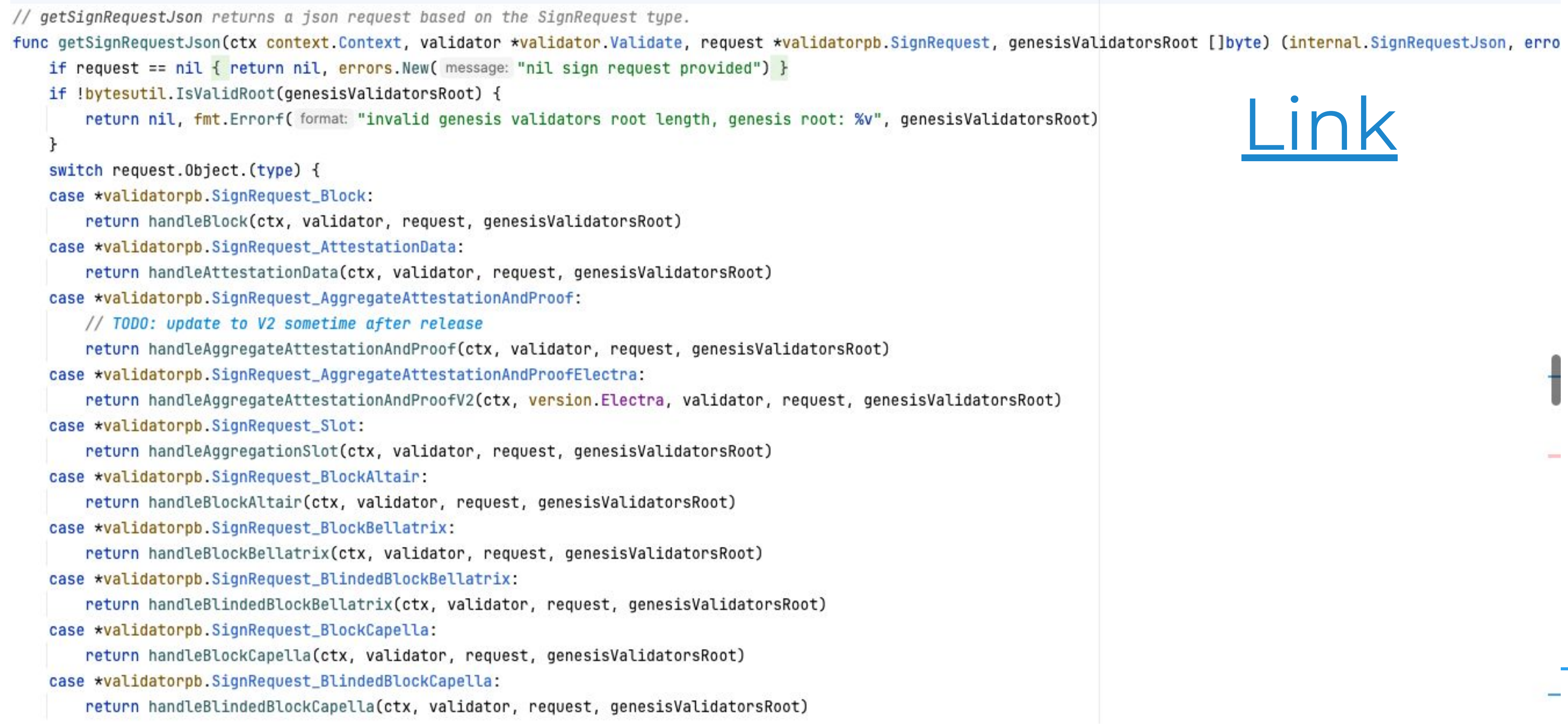
PRYSM
A Product of Offchain Labs

Wallet (remote keymanager)



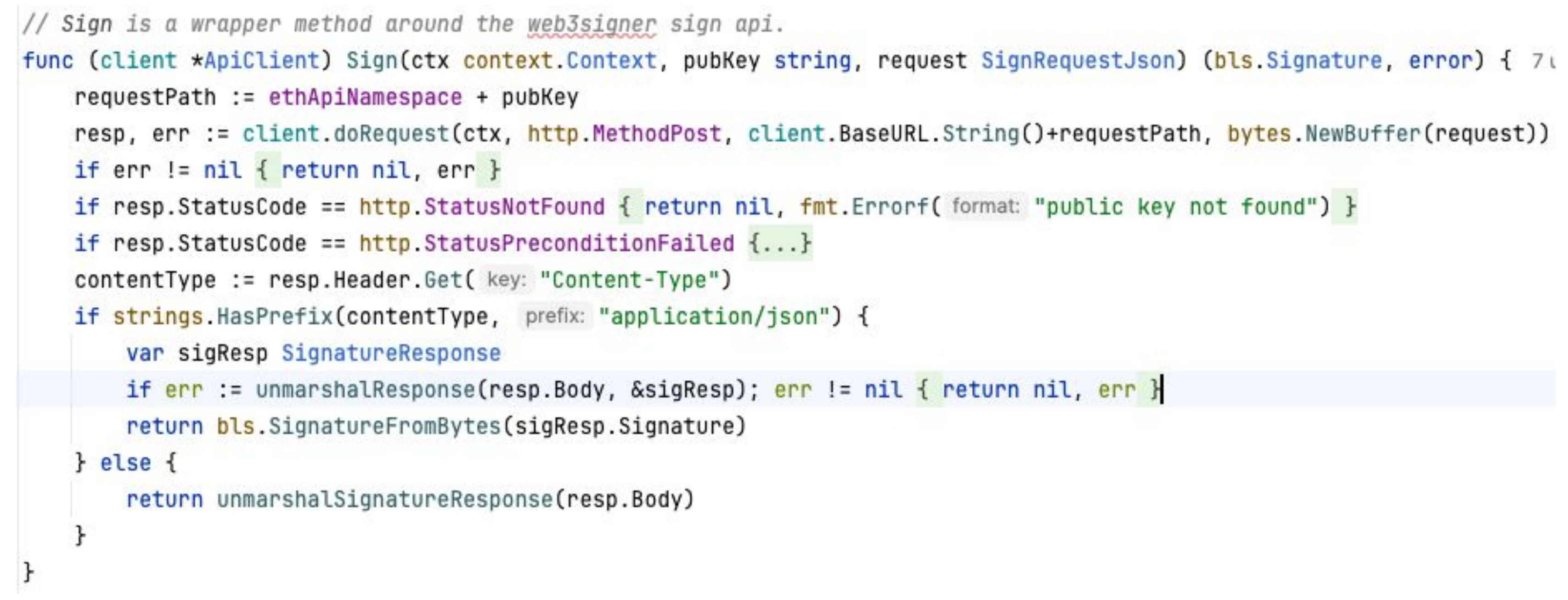


```go
// Sign signs the message by using a remote web3signer server.
func (km *Keymanager) Sign(ctx context.Context, request *validatorpb.SignRequest) (bls.Signature, error) {
    signRequest, err := getSignRequestJson(ctx, km.validator, request, km.genesisValidatorsRoot)
    if err != nil {
        erroredResponsesTotal.Inc()
        return nil, err
    }
    signature, err := km.client.Sign(ctx, hexutil.Encode(request.PublicKey), signRequest)
    if err != nil {
        erroredResponsesTotal.Inc()
        return nil, errors.Wrap(err,  message: "failed to sign the request")
    }
    log.WithField( key: "publicKey", request.PublicKey).Debug( args...: "Successfully signed the request")
    signRequestsTotal.Inc()
    return signature, nil
}
```



```go
// getSignRequestJson returns a json request based on the SignRequest type.
func getSignRequestJson(ctx context.Context, validator *validator.Validate, request *validatorpb.SignRequest, genesisValidatorsRoot []byte) (internal.SignRequestJson, erro
    if request == nil { return nil, errors.New( message: "nil sign request provided") }
    if !bytesutil.IsValidRoot(genesisValidatorsRoot) {
        return nil, fmt.Errorf( format: "invalid genesis validators root length, genesis root: %v", genesisValidatorsRoot)
    }
    switch request.Object.(type) {
    case *validatorpb.SignRequest_Block:
        return handleBlock(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_AttestationData:
        return handleAttestationData(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_AggregateAttestationAndProof:
        // TODO: update to V2 sometime after release
        return handleAggregateAttestationAndProof(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_AggregateAttestationAndProofElectra:
        return handleAggregateAttestationAndProofV2(ctx, version.Electra, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_Slot:
        return handleAggregationSlot(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_BlockAltair:
        return handleBlockAltair(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_BlockBellatrix:
        return handleBlockBellatrix(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_BlindedBlockBellatrix:
        return handleBlindedBlockBellatrix(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_BlockCapella:
        return handleBlockCapella(ctx, validator, request, genesisValidatorsRoot)
    case *validatorpb.SignRequest_BlindedBlockCapella:
        return handleBlindedBlockCapella(ctx, validator, request, genesisValidatorsRoot)
```

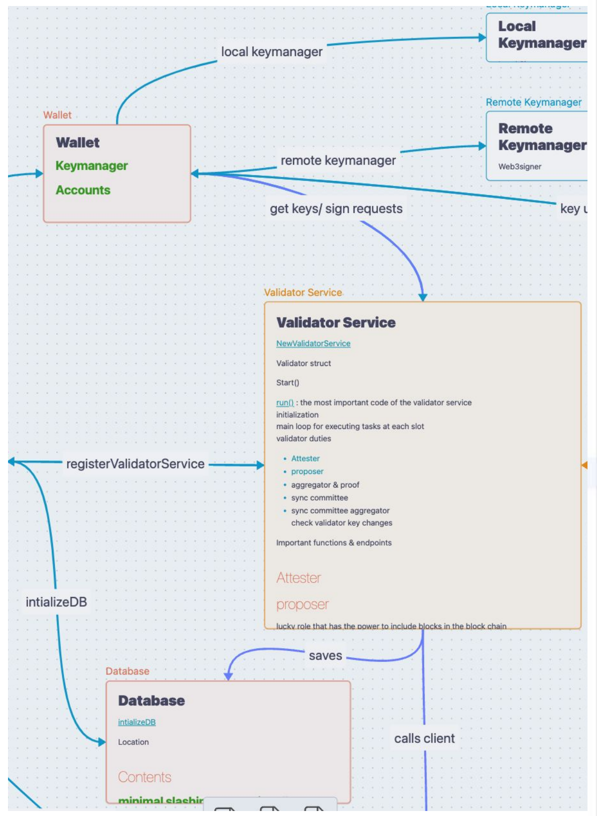
Link

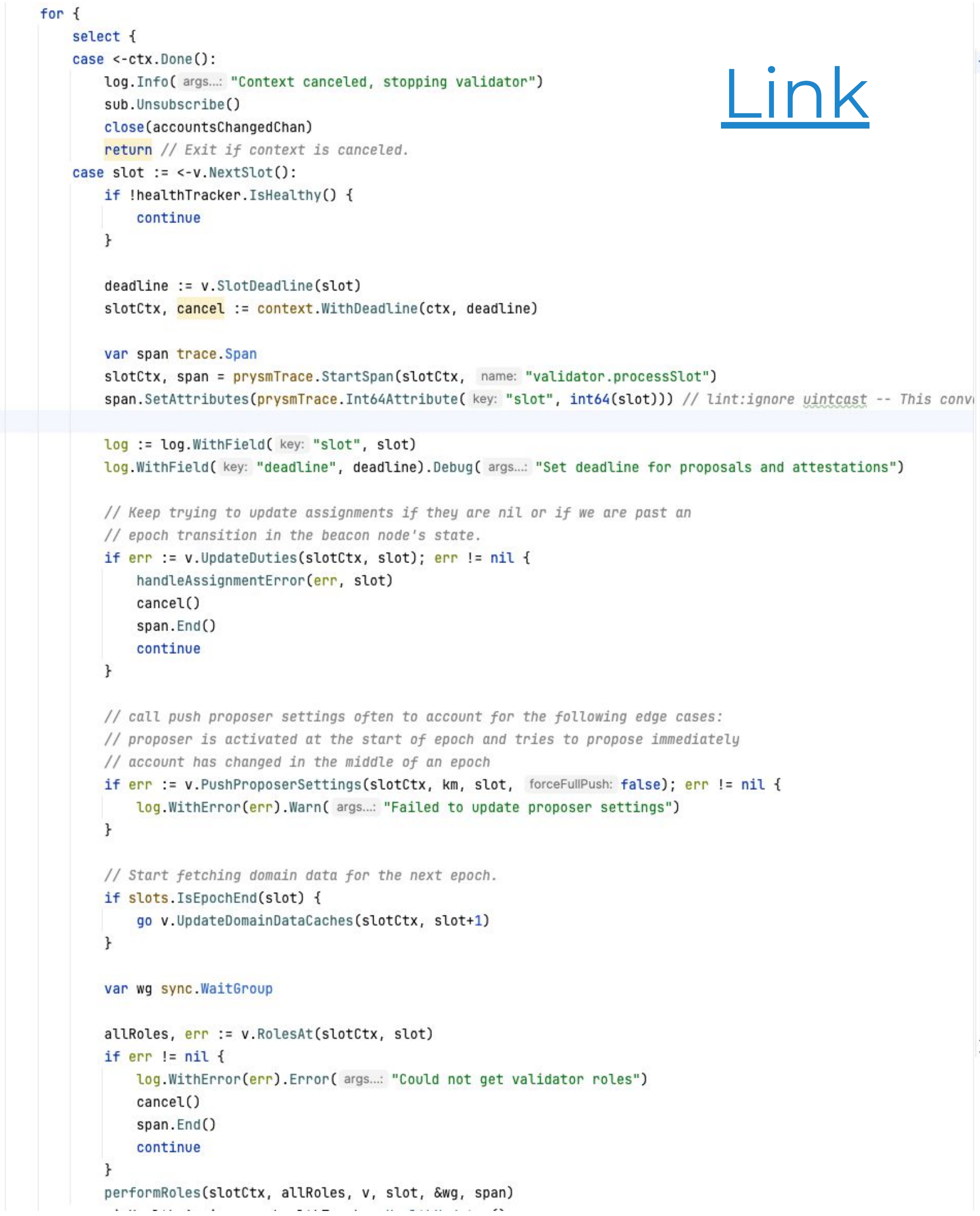


```go
// Sign is a wrapper method around the web3signer sign api.
func (client *ApiClient) Sign(ctx context.Context, pubKey string, request SignRequestJson) (bls.Signature, error) { 7
    requestPath := ethApiNamespace + pubKey
    resp, err := client.doRequest(ctx, http.MethodPost, client.BaseURL.String()+requestPath, bytes.NewBuffer(request))
    if err != nil { return nil, err }
    if resp.StatusCode == http.StatusNotFound { return nil, fmt.Errorf( format: "public key not found") }
    if resp.StatusCode == http.StatusPreconditionFailed {...}
    contentType := resp.Header.Get( key: "Content-Type")
    if strings.HasPrefix(contentType,  prefix: "application/json") {
        var sigResp SignatureResponse
        if err := unmarshalResponse(resp.Body, &sigResp); err != nil { return nil, err }
        return bls.SignatureFromBytes(sigResp.Signature)
    } else {
        return unmarshalSignatureResponse(resp.Body)
    }
}
```

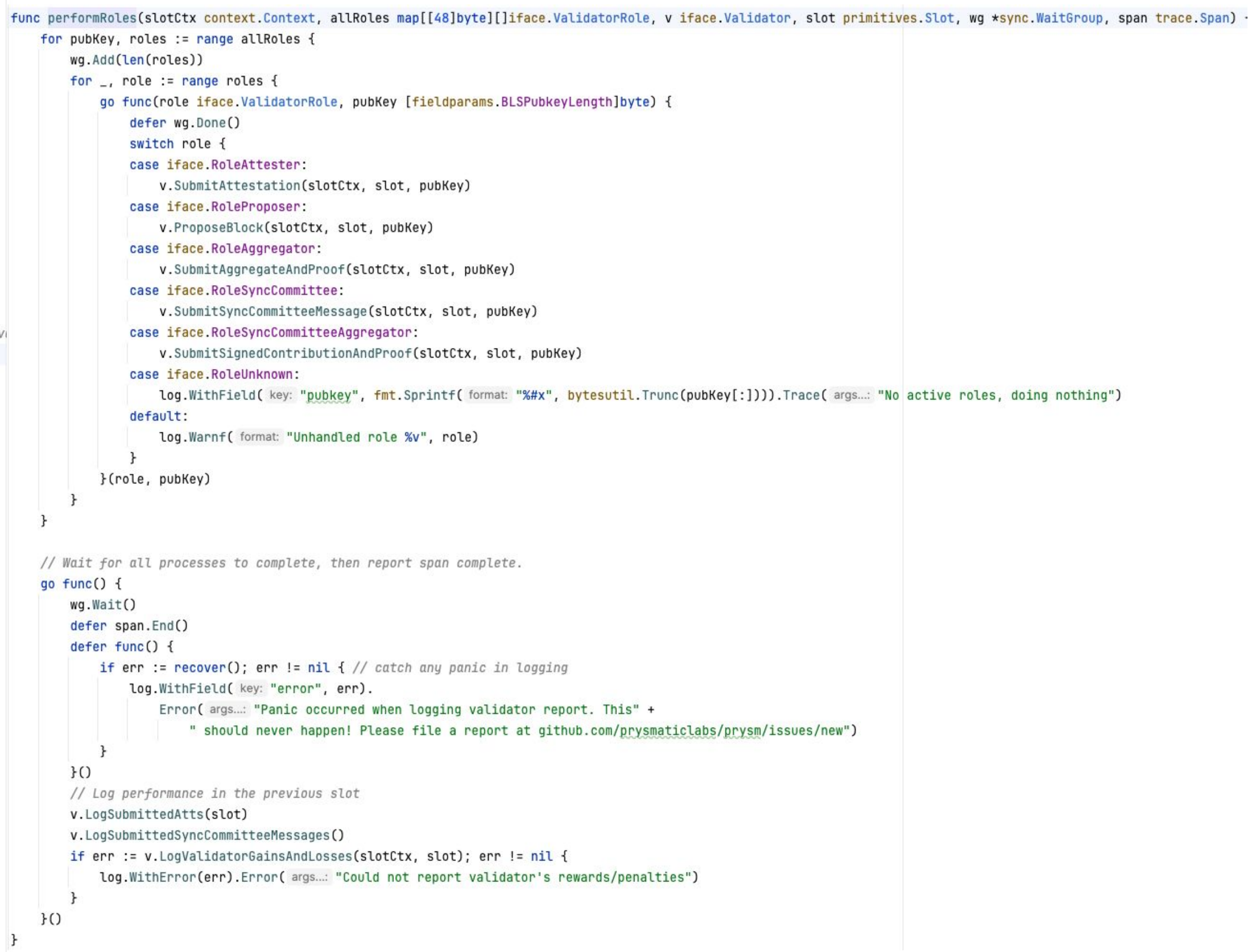
Validator Service continued …

Link





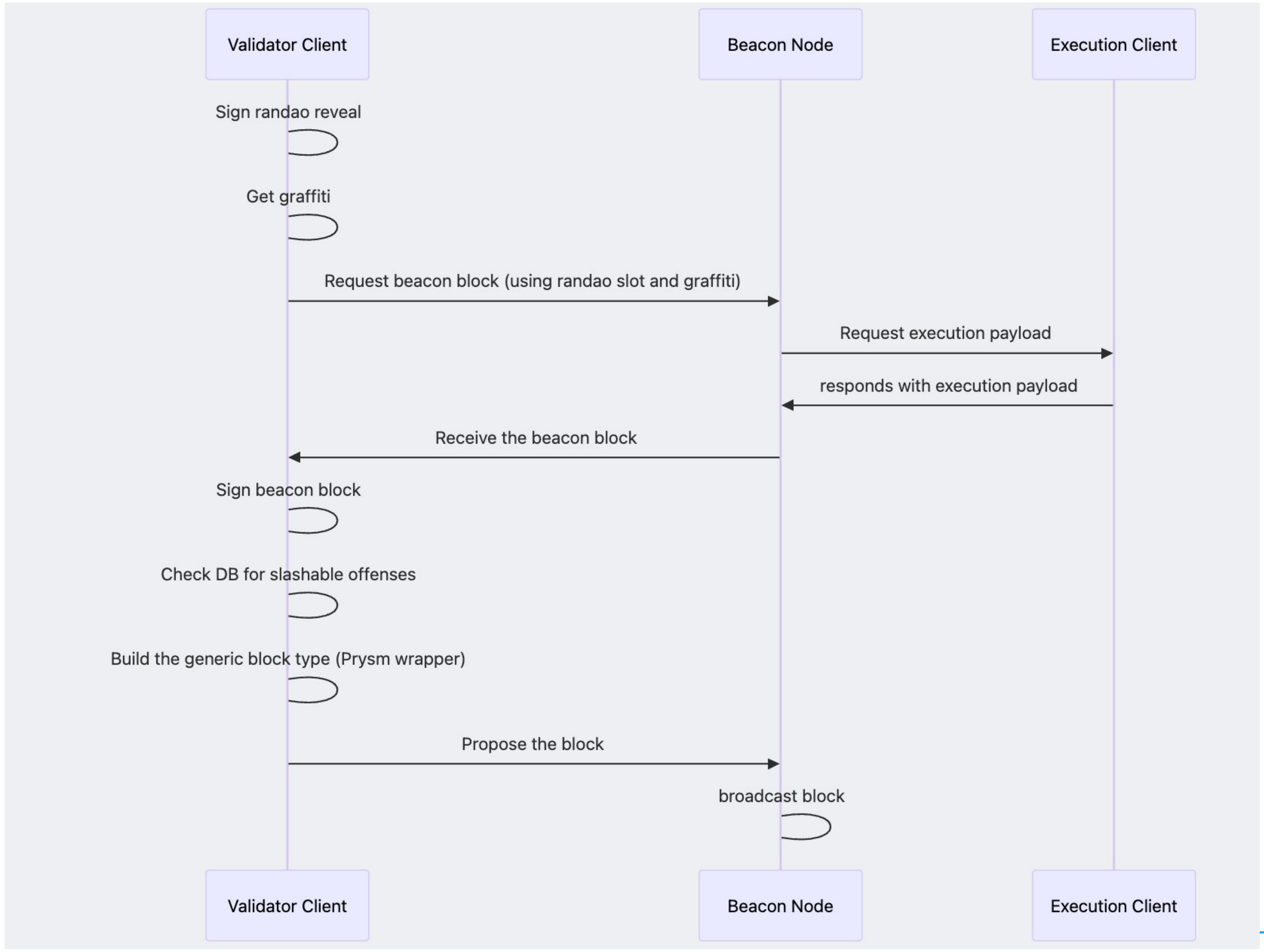
```go
for {
    select {
    case <-ctx.Done():
        log.Info( args...: "Context canceled, stopping validator")
        sub.Unsubscribe()
        close(accountsChangedChan)
        return // Exit if context is canceled.
    case slot := <-v.NextSlot():
        if !healthTracker.IsHealthy() {
            continue
        }

        deadline := v.SlotDeadline(slot)
        slotCtx, cancel := context.WithDeadline(ctx, deadline)

        var span trace.Span
        slotCtx, span = prysmTrace.StartSpan(slotCtx,  name: "validator.processSlot")
        span.SetAttributes(prysmTrace.Int64Attribute( key: "slot", int64(slot))) // lint:ignore uintcast -- This conv

        log := log.WithField( key: "slot", slot)
        log.WithField( key: "deadline", deadline).Debug( args...: "Set deadline for proposals and attestations")

        // Keep trying to update assignments if they are nil or if we are past an
        // epoch transition in the beacon node's state.
        if err := v.UpdateDuties(slotCtx, slot); err != nil {
            handleAssignmentError(err, slot)
            cancel()
            span.End()
            continue
        }

        // call push proposer settings often to account for the following edge cases:
        // proposer is activated at the start of epoch and tries to propose immediately
        // account has changed in the middle of an epoch
        if err := v.PushProposerSettings(slotCtx, km, slot,  forceFullPush: false); err != nil {
            log.WithError(err).Warn( args...: "Failed to update proposer settings")
        }

        // Start fetching domain data for the next epoch.
        if slots.IsEpochEnd(slot) {
            go v.UpdateDomainDataCaches(slotCtx, slot+1)
        }

        var wg sync.WaitGroup

        allRoles, err := v.RolesAt(slotCtx, slot)
        if err != nil {
            log.WithError(err).Error( args...: "Could not get validator roles")
            cancel()
            span.End()
            continue
        }
        performRoles(slotCtx, allRoles, v, slot, &wg, span)
```



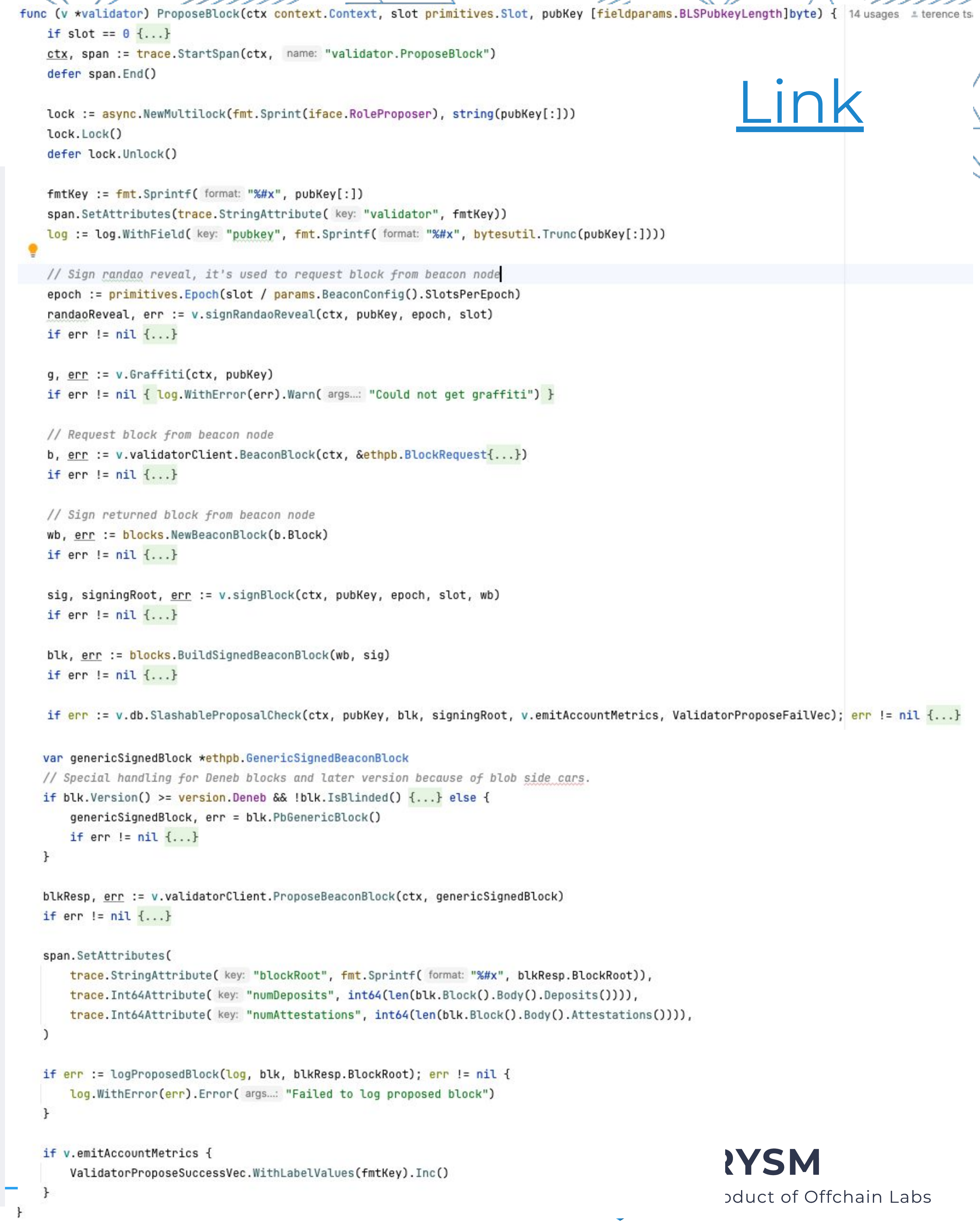
```go
func performRoles(slotCtx context.Context, allRoles map[[48]byte][]iface.ValidatorRole, v iface.Validator, slot primitives.Slot, wg *sync.WaitGroup, span trace.Span) ·
    for pubKey, roles := range allRoles {
        wg.Add(len(roles))
        for _, role := range roles {
            go func(role iface.ValidatorRole, pubKey [fieldparams.BLSPubkeyLength]byte) {
                defer wg.Done()
                switch role {
                case iface.RoleAttester:
                    v.SubmitAttestation(slotCtx, slot, pubKey)
                case iface.RoleProposer:
                    v.ProposeBlock(slotCtx, slot, pubKey)
                case iface.RoleAggregator:
                    v.SubmitAggregateAndProof(slotCtx, slot, pubKey)
                case iface.RoleSyncCommittee:
                    v.SubmitSyncCommitteeMessage(slotCtx, slot, pubKey)
                case iface.RoleSyncCommitteeAggregator:
                    v.SubmitSignedContributionAndProof(slotCtx, slot, pubKey)
                case iface.RoleUnknown:
                    log.WithField( key: "pubkey", fmt.Sprintf( format: "%#x", bytesutil.Trunc(pubKey[:]))).Trace( args...: "No active roles, doing nothing")
                default:
                    log.Warnf( format: "Unhandled role %v", role)
                }
            }(role, pubKey)
        }
    }

    // Wait for all processes to complete, then report span complete.
    go func() {
        wg.Wait()
        defer span.End()
        defer func() {
            if err := recover(); err != nil { // catch any panic in logging
                log.WithField( key: "error", err).
                    Error( args...: "Panic occurred when logging validator report. This" +
                        " should never happen! Please file a report at github.com/prysmaticlabs/prysm/issues/new")
            }
        }()
        // Log performance in the previous slot
        v.LogSubmittedAtts(slot)
        v.LogSubmittedSyncCommitteeMessages()
        if err := v.LogValidatorGainsAndLosses(slotCtx, slot); err != nil {
            log.WithError(err).Error( args...: "Could not report validator's rewards/penalties")
        }
    }()
}
```

PRYSM
A Product of Offchain Labs

Propose Block



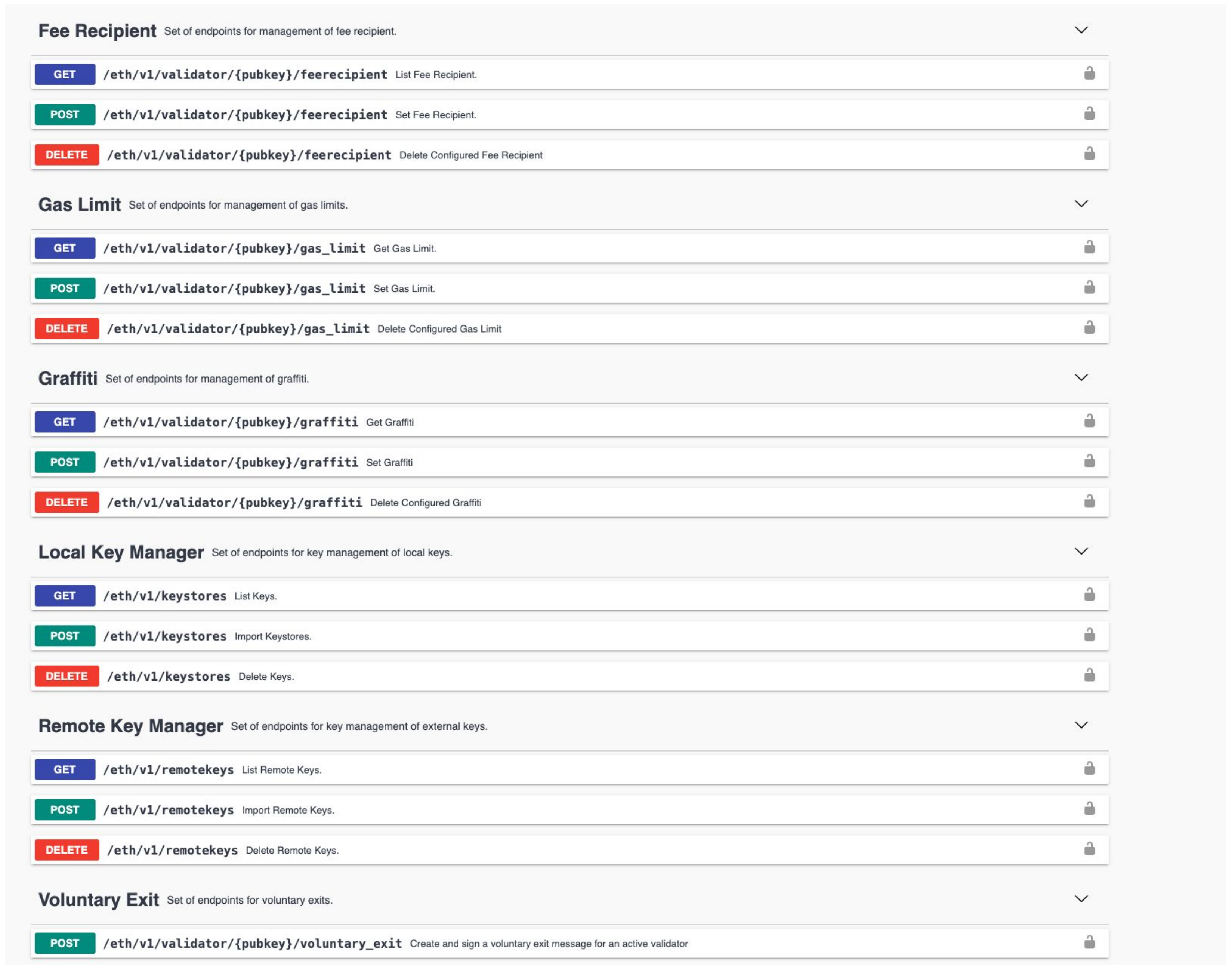
Link



```
func (v *validator) ProposeBlock(ctx context.Context, slot primitives.Slot, pubKey [fieldparams.BLSPubkeyLength]byte) {   14 usages   terence ts
    if slot == 0 {...}
    ctx, span := trace.StartSpan(ctx,  name: "validator.ProposeBlock")
    defer span.End()

    lock := async.NewMultilock(fmt.Sprint(iface.RoleProposer), string(pubKey[:]))
    lock.Lock()
    defer lock.Unlock()

    fmtKey := fmt.Sprintf( format: "%#x", pubKey[:])
    span.SetAttributes(trace.StringAttribute( key: "validator", fmtKey))
    log := log.WithField( key: "pubkey", fmt.Sprintf( format: "%#x", bytesutil.Trunc(pubKey[:])))

    // Sign randao reveal, it's used to request block from beacon node
    epoch := primitives.Epoch(slot / params.BeaconConfig().SlotsPerEpoch)
    randaoReveal, err := v.signRandaoReveal(ctx, pubKey, epoch, slot)
    if err != nil {...}

    g, err := v.Graffiti(ctx, pubKey)
    if err != nil { log.WithError(err).Warn( args...: "Could not get graffiti") }

    // Request block from beacon node
    b, err := v.validatorClient.BeaconBlock(ctx, &ethpb.BlockRequest{...})
    if err != nil {...}

    // Sign returned block from beacon node
    wb, err := blocks.NewBeaconBlock(b.Block)
    if err != nil {...}

    sig, signingRoot, err := v.signBlock(ctx, pubKey, epoch, slot, wb)
    if err != nil {...}

    blk, err := blocks.BuildSignedBeaconBlock(wb, sig)
    if err != nil {...}

    if err := v.db.SlashableProposalCheck(ctx, pubKey, blk, signingRoot, v.emitAccountMetrics, ValidatorProposeFailVec); err != nil {...}

   var genericSignedBlock *ethpb.GenericSignedBeaconBlock
   // Special handling for Deneb blocks and later version because of blob side cars.
   if blk.Version() >= version.Deneb && !blk.IsBlinded() {...} else {
       genericSignedBlock, err = blk.PbGenericBlock()
       if err != nil {...}
   }

   blkResp, err := v.validatorClient.ProposeBeaconBlock(ctx, genericSignedBlock)
   if err != nil {...}

   span.SetAttributes(
       trace.StringAttribute( key: "blockRoot", fmt.Sprintf( format: "%#x", blkResp.BlockRoot)),
       trace.Int64Attribute( key: "numDeposits", int64(len(blk.Block().Body().Deposits()))),
       trace.Int64Attribute( key: "numAttestations", int64(len(blk.Block().Body().Attestations()))),
   )

   if err := logProposedBlock(log, blk, blkResp.BlockRoot); err != nil {
       log.WithError(err).Error( args...: "Failed to log proposed block")
   }

   if v.emitAccountMetrics {
       ValidatorProposeSuccessVec.WithLabelValues(fmtKey).Inc()
   }
}
```

Keymanager APIs

https://ethereum.github.io/keymanager-APIs/



Fee Recipient Set of endpoints for management of fee recipient.

- GET `/eth/v1/validator/{pubkey}/feerecipient` List Fee Recipient.
- POST `/eth/v1/validator/{pubkey}/feerecipient` Set Fee Recipient.
- DELETE `/eth/v1/validator/{pubkey}/feerecipient` Delete Configured Fee Recipient

Gas Limit Set of endpoints for management of gas limits.

- GET `/eth/v1/validator/{pubkey}/gas_limit` Get Gas Limit.
- POST `/eth/v1/validator/{pubkey}/gas_limit` Set Gas Limit.
- DELETE `/eth/v1/validator/{pubkey}/gas_limit` Delete Configured Gas Limit

Graffiti Set of endpoints for management of graffiti.

- GET `/eth/v1/validator/{pubkey}/graffiti` Get Graffiti
- POST `/eth/v1/validator/{pubkey}/graffiti` Set Graffiti
- DELETE `/eth/v1/validator/{pubkey}/graffiti` Delete Configured Graffiti

Local Key Manager Set of endpoints for key management of local keys.

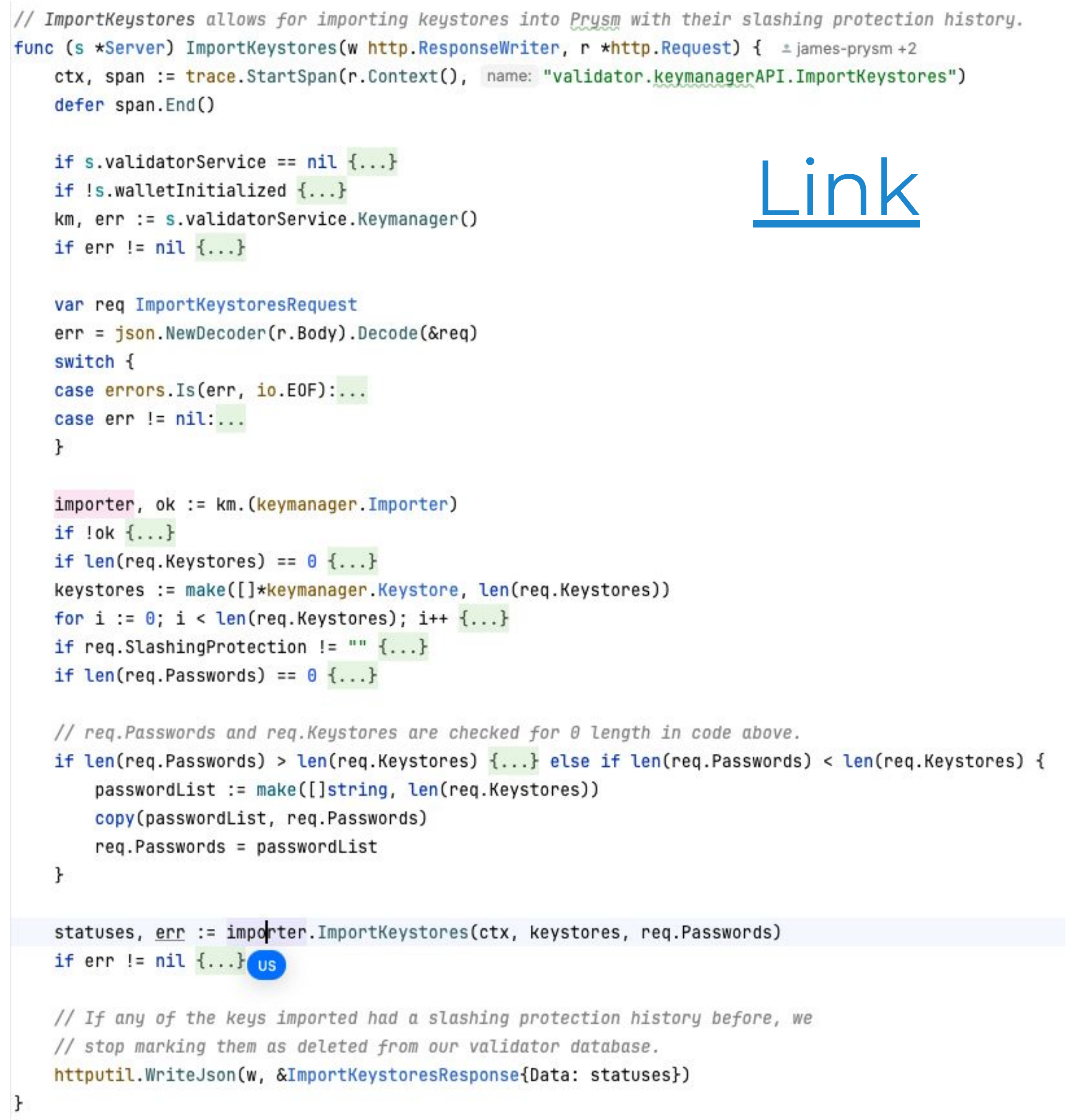
- GET `/eth/v1/keystores` List Keys.
- POST `/eth/v1/keystores` Import Keystores.
- DELETE `/eth/v1/keystores` Delete Keys.

Remote Key Manager Set of endpoints for key management of external keys.

- GET `/eth/v1/remotekeys` List Remote Keys.
- POST `/eth/v1/remotekeys` Import Remote Keys.
- DELETE `/eth/v1/remotekeys` Delete Remote Keys.

Voluntary Exit Set of endpoints for voluntary exits.

- POST `/eth/v1/validator/{pubkey}/voluntary_exit` Create and sign a voluntary exit message for an active validator

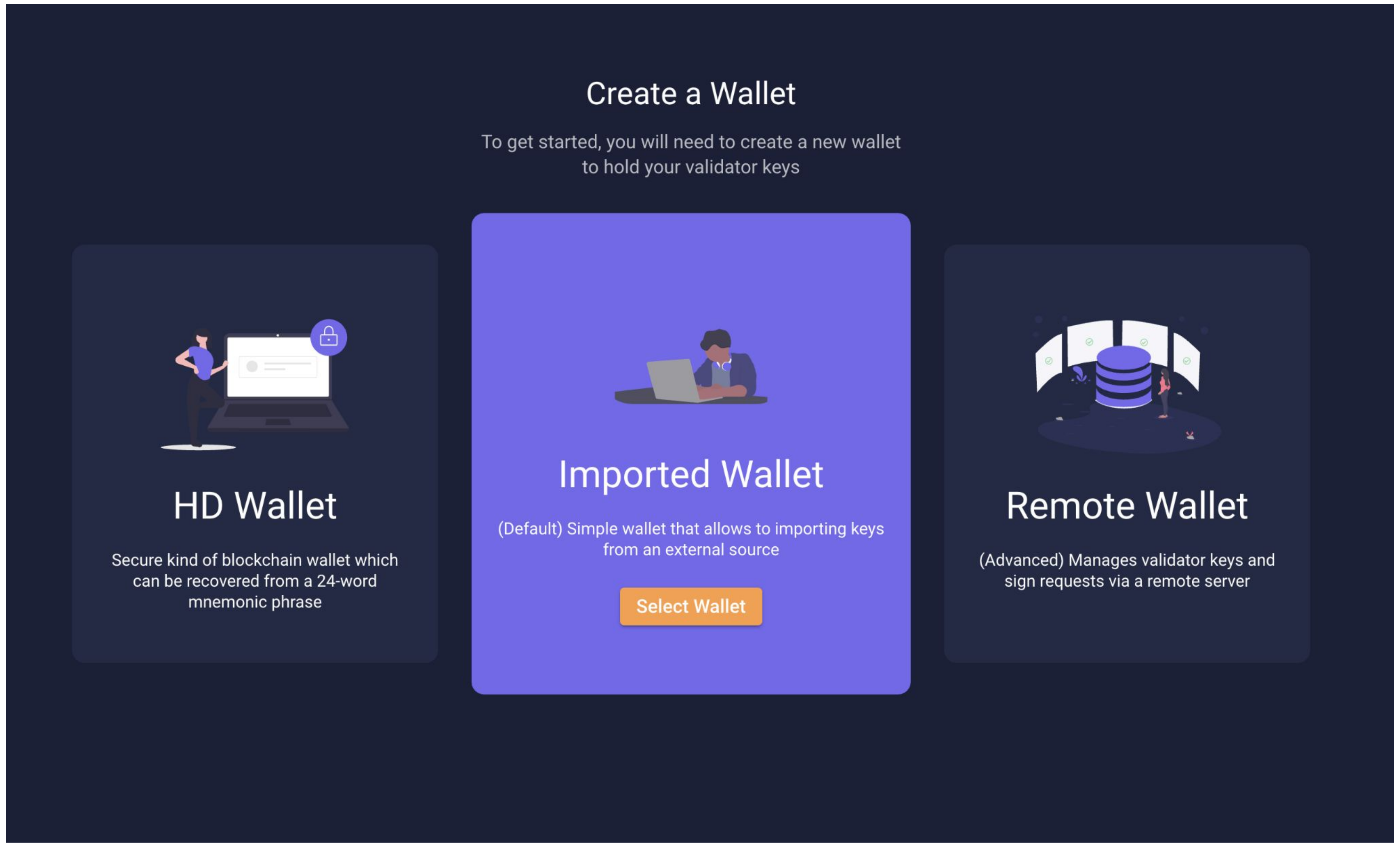


```go
// ImportKeystores allows for importing keystores into Prysm with their slashing protection history.
func (s *Server) ImportKeystores(w http.ResponseWriter, r *http.Request) {  james-prysm +2
    ctx, span := trace.StartSpan(r.Context(),  name: "validator.keymanagerAPI.ImportKeystores")
    defer span.End()

    if s.validatorService == nil {...}
    if !s.walletInitialized {...}
    km, err := s.validatorService.Keymanager()
    if err != nil {...}

    var req ImportKeystoresRequest
    err = json.NewDecoder(r.Body).Decode(&req)
    switch {
    case errors.Is(err, io.EOF):...
    case err != nil:...
    }

    importer, ok := km.(keymanager.Importer)
    if !ok {...}
    if len(req.Keystores) == 0 {...}
    keystores := make([]*keymanager.Keystore, len(req.Keystores))
    for i := 0; i < len(req.Keystores); i++ {...}
    if req.SlashingProtection != "" {...}
    if len(req.Passwords) == 0 {...}

    // req.Passwords and req.Keystores are checked for 0 length in code above.
    if len(req.Passwords) > len(req.Keystores) {...} else if len(req.Passwords) < len(req.Keystores) {
        passwordList := make([]string, len(req.Keystores))
        copy(passwordList, req.Passwords)
        req.Passwords = passwordList
    }

    statuses, err := importer.ImportKeystores(ctx, keystores, req.Passwords)
    if err != nil {...}

    // If any of the keys imported had a slashing protection history before, we
    // stop marking them as deleted from our validator database.
    httputil.WriteJson(w, &ImportKeystoresResponse{Data: statuses})
}
```

Link

PRYSM
A Product of Offchain Labs

WebUI


Create a Wallet

To get started, you will need to create a new wallet to hold your validator keys

HD Wallet

Secure kind of blockchain wallet which can be recovered from a 24-word mnemonic phrase

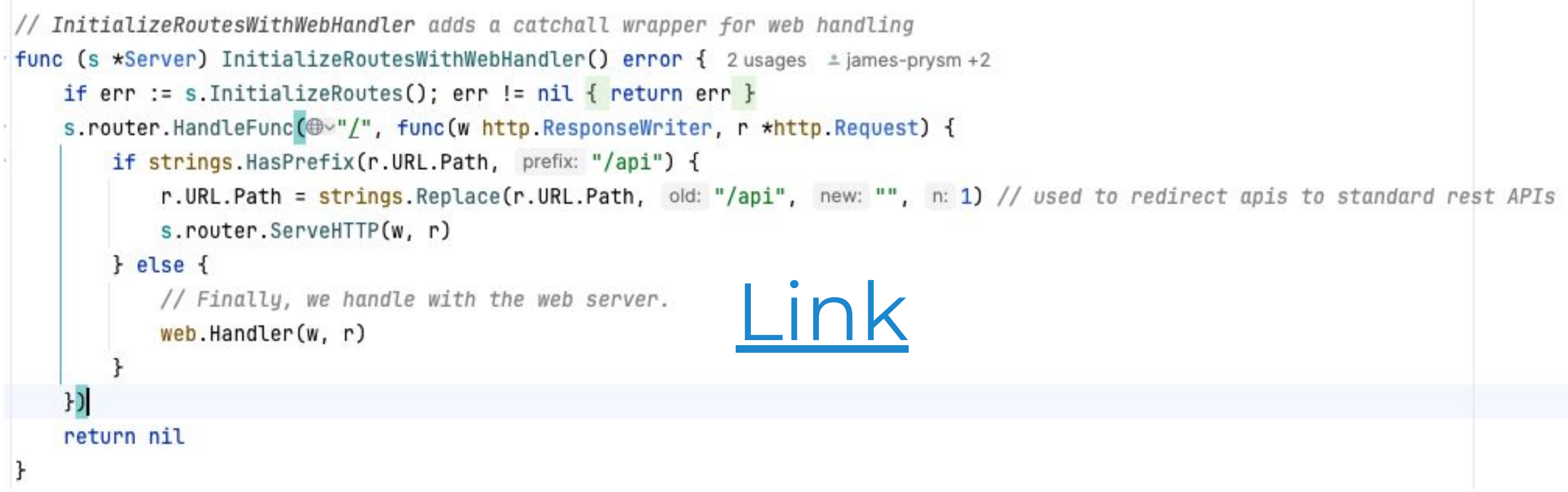
Imported Wallet

(Default) Simple wallet that allows to importing keys from an external source

Select Wallet

Remote Wallet

(Advanced) Manages validator keys and sign requests via a remote server

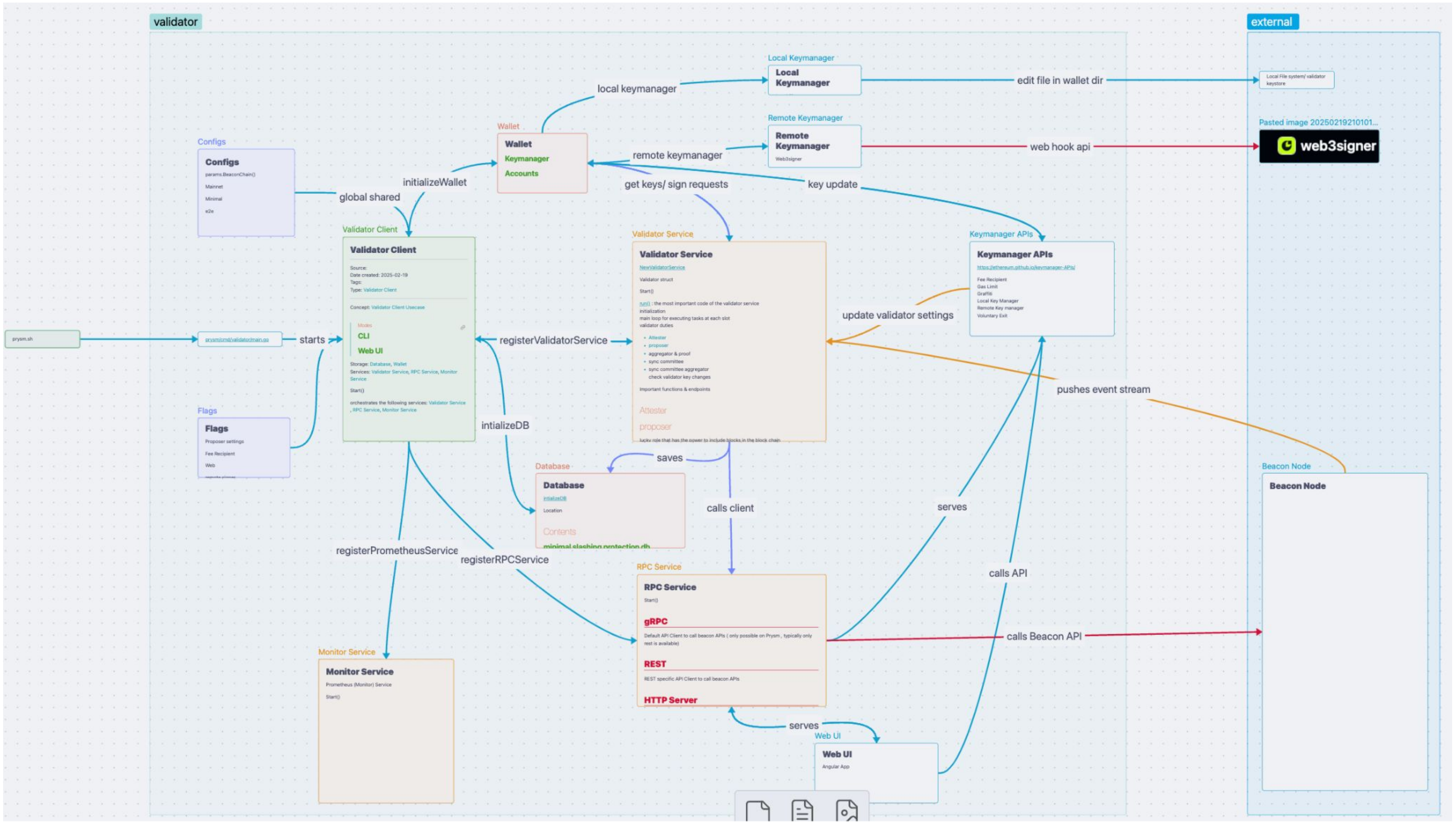

```go
// InitializeRoutesWithWebHandler adds a catchall wrapper for web handling
func (s *Server) InitializeRoutesWithWebHandler() error {  2 usages  james-prysm +2
    if err := s.InitializeRoutes(); err != nil { return err }
    s.router.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        if strings.HasPrefix(r.URL.Path, prefix: "/api") {
            r.URL.Path = strings.Replace(r.URL.Path, old: "/api", new: "", n: 1) // used to redirect apis to standard rest APIs
            s.router.ServeHTTP(w, r)
        } else {
            // Finally, we handle with the web server.
            web.Handler(w, r)
        }
    })
    return nil
}
```

Link

https://github.com/prysmaticlabs/prysm-web-ui

PRYSM
A Product of Offchain Labs

Review of architecture



1. **Manage/Use** your validator **keystores**

2. Perform validator **protocol duties**

PRYSM
A Product of Offchain Labs