readme.md for @push.rocks/smartserve

A blazing-fast, cross-platform HTTP server for Node.js, Deno, and Bun with decorator-based routing, OpenAPI/Swagger integration, automatic compression, WebSocket support, static file serving, and WebDAV protocol. □□

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

```
npm install @push.rocks/smartserve
# or
pnpm add @push.rocks/smartserve
```

Features

Feature	Description
□ **Cross-Platform**	Works seamlessly on Node.js, Deno, and Bun with zero config
□□ **Decorator-Based Routing**	Clean, expressive `@Route`, `@Get`, `@Post` decorators
□□ **OpenAPI/Swagger**	Auto-generate OpenAPI 3.1 specs with built-in Swagger UI & ReDoc

Feature	Description
□ **Request Validation**	Validate requests against JSON Schema with automatic coercion
□□ **Auto Compression**	Brotli/gzip compression with smart content detection
□□ **Guards & Interceptors**	Built-in `@Guard`, `@Transform`, `@Intercept` for auth & transformation
□□ **Static File Server**	Streaming, ETags, Range requests, directory listing, pre-compressed files
□□ **WebDAV Support**	Mount as network drive with full RFC 4918 compliance
□□ **WebSocket Ready**	Native WebSocket support with TypedRouter for type-safe RPC
⚡ **Zero Overhead**	Native Web Standards API (Request/Response) on Deno/Bun
□□ **HTTPS/TLS**	Built-in TLS support with certificate configuration

Quick Start

```
import { SmartServe, Route, Get, Post, type IRequestContext } from '@push.rocks/smartserve';

@Route('/api')
class UserController {
  @Get('/hello')
  hello() {
    return { message: 'Hello World! □□' };
  }

  @Get('/users/:id')
  getUser(ctx: IRequestContext) {
    return { id: ctx.params.id, name: 'John Doe' };
  }

  @Post('/users')
  async createUser(ctx: IRequestContext<{ name: string; email: string }>) {
    const body = await ctx.json();
    return { id: 'new-id', ...body };
  }
}
```

```
const server = new SmartServe({ port: 3000 });
server.register(UserController);
await server.start();

console.log('□□Server running at http://localhost:3000');
```

Table of Contents

Decorators

Route Decorators

```
import { Route, Get, Post, Put, Delete, Patch, All } from '@push.rocks/smartserve';

@Route('/api/v1')  // Base path for all routes in this controller
class ApiController {
  @Get('/items')           // GET /api/v1/items
  listItems() {
    return [{ id: 1, name: 'Item 1' }];
  }

  @Get('/items/:id')       // GET /api/v1/items/:id
  getItem(ctx: IRequestContext) {
    return { id: ctx.params.id };
  }

  @Post('/items')          // POST /api/v1/items
  async createItem(ctx: IRequestContext<{ name: string }>) {
    const body = await ctx.json();
    return { created: body.name };
  }

  @Put('/items/:id')       // PUT /api/v1/items/:id
  async updateItem(ctx: IRequestContext) {
    const body = await ctx.json();
    return { updated: ctx.params.id, ...body };
  }

  @Delete('/items/:id')    // DELETE /api/v1/items/:id
  deleteItem(ctx: IRequestContext) {
    return { deleted: ctx.params.id };
  }

  @All('/webhook')         // Matches ALL HTTP methods
  handleWebhook(ctx: IRequestContext) {
    return { method: ctx.method };
  }
}
```

Guards (Authentication/Authorization)

Guards protect routes by returning `true` (allow) or `false` (reject with 403):

```
import { Route, Get, Guard, hasBearerToken, type IRequestContext } from
'@push.rocks/smartserve';

// Custom guard function
const isAuthenticated = (ctx: IRequestContext) => {
  return ctx.headers.has('Authorization');
};

const isAdmin = (ctx: IRequestContext) => {
  return ctx.headers.get('X-Role') === 'admin';
};

@Route('/admin')
@Guard(isAuthenticated)
@Guard(isAdmin)  // Multiple guards - all must pass
class AdminController {
  @Get('/dashboard')
  dashboard() {
    return { admin: true };
  }

  // Method-level guard (runs after class guards)
  @Get('/super-secret')
  @Guard((ctx) => ctx.headers.get('X-Super') === 'yes')
  superSecret() {
    return { level: 'super-secret' };
  }
}

// Built-in utility guards
@Route('/protected')
@Guard(hasBearerToken())  // Requires Authorization: Bearer <token>
class ProtectedController {
  @Get('/data')
  getData() {
    return { protected: true };
  }
}
```

Transforms (Response Modification)

Transforms modify the response before sending:

```
import { Route, Get, Transform, wrapSuccess, addTimestamp } from '@push.rocks/smartserve';

// Custom transform
const addVersion = <T extends object>(data: T) => ({
  ...data,
  apiVersion: '2.0',
});

@Route('/api')
@Transform(wrapSuccess)  // Built-in: wraps in { success: true, data: ... }
class ApiController {
  @Get('/info')
  @Transform(addTimestamp)  // Built-in: adds timestamp field
  @Transform(addVersion)    // Transforms stack
  getInfo() {
    return { name: 'MyAPI' };
  }
  // Response: { success: true, data: { name: 'MyAPI', timestamp: '...', apiVersion: '2.0' } }
}
```

Intercept (Full Control)

For complete control over request/response flow:

```
import { Route, Get, Intercept, type IRequestContext } from '@push.rocks/smartserve';

@Route('/api')
@Intercept({
  // Runs BEFORE handler
  request: async (ctx) => {
    console.log(`□□${ctx.method} ${ctx.path}`);

    // Return Response to short-circuit
    if (ctx.headers.get('X-Block') === 'true') {
      return new Response('Blocked', { status: 403 });
```

```
    }

    // Add data to state for handler access
    ctx.state.requestTime = Date.now();

    // Return void to continue with original context
  },

  // Runs AFTER handler
  response: async (data, ctx) => {
    const duration = Date.now() - (ctx.state.requestTime as number);
    console.log(`□□Response in ${duration}ms`);
    return { ...data, processedIn: `${duration}ms` };
  },
})
class LoggedController {
  @Get('/data')
  getData() {
    return { items: [1, 2, 3] };
  }
}
```

OpenAPI & Swagger

SmartServe includes first-class OpenAPI 3.1 support with automatic spec generation, Swagger UI, ReDoc, and request validation.

Documenting APIs

```
import {
  SmartServe,
  Route,
  Get,
  Post,
  ApiOperation,
  ApiParam,
```

```
  ApiQuery,
  ApiRequestBody,
  ApiResponseBody,
  ApiTag,
  ApiSecurity,
  type IRequestContext,
} from '@push.rocks/smartserve';

// Define JSON Schemas for validation
const UserSchema = {
  type: 'object',
  properties: {
    id: { type: 'string', format: 'uuid' },
    name: { type: 'string', minLength: 1 },
    email: { type: 'string', format: 'email' },
  },
  required: ['id', 'name', 'email'],
} as const;

const CreateUserSchema = {
  type: 'object',
  properties: {
    name: { type: 'string', minLength: 1 },
    email: { type: 'string', format: 'email' },
  },
  required: ['name', 'email'],
} as const;

@Route('/api/users')
@ApiTag('Users')
class UserController {
  @Get('/')
  @ApiOperation({
    summary: 'List all users',
    description: 'Returns a paginated list of users',
  })
  @ApiQuery('page', {
    description: 'Page number',
    schema: { type: 'integer', minimum: 1, default: 1 },
```

```
})
@ApiQuery('limit', {
  description: 'Items per page',
  schema: { type: 'integer', minimum: 1, maximum: 100, default: 20 },
})
@ApiResponseBody(200, {
  description: 'List of users',
  schema: { type: 'array', items: UserSchema },
})
listUsers(ctx: IRequestContext) {
  const page = ctx.query.page ?? '1';
  const limit = ctx.query.limit ?? '20';
  return { users: [], page: parseInt(page), limit: parseInt(limit) };
}

@Get('/:id')
@ApiOperation({ summary: 'Get user by ID' })
@ApiParam('id', {
  description: 'User UUID',
  schema: { type: 'string', format: 'uuid' },
})
@ApiResponseBody(200, { description: 'User found', schema: UserSchema })
@ApiResponseBody(404, { description: 'User not found' })
getUser(ctx: IRequestContext) {
  return { id: ctx.params.id, name: 'John Doe', email: 'john@example.com' };
}

@Post('/')
@ApiOperation({ summary: 'Create a new user' })
@ApiRequestBody({
  description: 'User data',
  schema: CreateUserSchema,
})
@ApiResponseBody(201, { description: 'User created', schema: UserSchema })
@ApiResponseBody(400, { description: 'Validation error' })
@ApiSecurity('bearerAuth')
async createUser(ctx: IRequestContext<{ name: string; email: string }>) {
  const body = await ctx.json();
  return { id: 'new-uuid', name: body.name, email: body.email };
```

```
  }
}
```

Request Validation

When you define `@ApiRequestBody`, `@ApiParam`, or `@ApiQuery` with schemas, SmartServe **automatically validates** incoming requests:

```
const server = new SmartServe({
  port: 3000,
  openapi: {
    enabled: true,
    info: {
      title: 'My API',
      version: '1.0.0',
      description: 'A well-documented API',
    },
    validate: true,  // □□Enable automatic request validation
  },
});

server.register(UserController);
await server.start();

// Invalid request → 400 Bad Request with details
// POST /api/users with { "name": "" }
// Response: { "error": "Validation failed", "source": "body", "details": [...] }
```

Automatic Type Coercion: Query and path parameters are automatically coerced to their schema types:

```
@Get('/items')
@ApiQuery('page', { schema: { type: 'integer', default: 1 } })
@ApiQuery('active', { schema: { type: 'boolean' } })
listItems(ctx: IRequestContext) {
  // ctx.query.page is coerced to number (1)
  // ctx.query.active is coerced to boolean
  return { page: ctx.query.page, active: ctx.query.active };
}
```

Swagger UI & ReDoc

```
const server = new SmartServe({
  port: 3000,
  openapi: {
    enabled: true,
    info: {
      title: 'My Awesome API',
      version: '2.0.0',
      description: 'API documentation with interactive testing',
      contact: {
        name: 'API Support',
        email: 'support@example.com',
      },
    },
    servers: [
      { url: 'http://localhost:3000', description: 'Development' },
      { url: 'https://api.example.com', description: 'Production' },
    ],
    securitySchemes: {
      bearerAuth: {
        type: 'http',
        scheme: 'bearer',
        bearerFormat: 'JWT',
      },
    },
    // Customize paths
    specPath: '/openapi.json',    // Default: /openapi.json
    swaggerPath: '/docs',         // Default: /docs
    redocPath: '/redoc',          // Default: /redoc
  },
});

await server.start();

// □□Swagger UI:  http://localhost:3000/docs
// □□ReDoc:       http://localhost:3000/redoc
// □□OpenAPI:     http://localhost:3000/openapi.json
```

Compression

SmartServe automatically compresses responses using Brotli or gzip based on client support:

```
const server = new SmartServe({
  port: 3000,
  compression: {
    enabled: true,          // Default: true
    threshold: 1024,        // Min bytes to compress (default: 1KB)
    level: 6,               // Compression level 1-11 for br, 1-9 for gzip
    preferBrotli: true,     // Prefer Brotli over gzip
  },
});
```

Per-Route Compression Control

```
import { Route, Get, Compress, NoCompress } from '@push.rocks/smartserve';

@Route('/api')
class ApiController {
  @Get('/large-data')
  @Compress({ level: 9 })  // Force high compression
  getLargeData() {
    return { data: '...massive payload...' };
  }

  @Get('/already-compressed')
  @NoCompress()  // Skip compression (e.g., for pre-compressed content)
  getCompressed() {
    return someCompressedBuffer;
  }
}
```

Pre-Compressed Static Files

Serve `.br` or `.gz` files automatically when available:

```
const server = new SmartServe({
  port: 3000,
  static: {
    root: './dist',
    precompressed: true,  // Serve main.js.br instead of main.js
  },
});
```

Static File Server

Serve static files with streaming, ETags, Range requests, and directory listing:

```
const server = new SmartServe({
  port: 3000,
  static: {
    root: './public',
    index: ['index.html', 'index.htm'],
    dotFiles: 'deny',               // 'allow' | 'deny' | 'ignore'
    etag: true,                     // Generate ETags for caching
    lastModified: true,             // Add Last-Modified header
    cacheControl: 'max-age=3600',   // Or function: (path) => 'max-age=...'
    extensions: ['.html'],          // Try these extensions for extensionless URLs
    precompressed: true,            // Serve .br/.gz files when available
    directoryListing: {
      showHidden: false,
      sortBy: 'name',               // 'name' | 'size' | 'modified'
      sortOrder: 'asc',
    },
  },
});
```

Or use the shorthand:

```
const server = new SmartServe({
  port: 3000,
  static: './public',  // Uses sensible defaults
});
```

WebDAV Support

Mount the server as a network drive on macOS, Windows, or Linux:

```
const server = new SmartServe({
  port: 8080,
  webdav: {
    root: '/path/to/files',
    auth: (ctx) => {
      // Optional: Basic authentication
      const auth = ctx.headers.get('Authorization');
      if (!auth) return false;
      const [, credentials] = auth.split(' ');
      const [user, pass] = atob(credentials).split(':');
      return user === 'admin' && pass === 'secret';
    },
    locking: true,  // Enable RFC 4918 exclusive write locks
  },
});

await server.start();
// □□Connect: Finder → Go → Connect to Server → http://localhost:8080
```

Supported WebDAV Methods:

Method	Description
`OPTIONS`	Capability discovery
`PROPFIND`	Directory listing and file metadata
`MKCOL`	Create directory
`COPY`	Copy files/directories
`MOVE`	Move/rename files/directories
`LOCK`	Acquire exclusive write lock
`UNLOCK`	Release lock
`GET` / `PUT` / `DELETE`	File operations

WebSocket Support

WebSocket connections are handled natively across all runtimes:

```
const server = new SmartServe({
  port: 3000,
  websocket: {
    onOpen: (peer) => {
      console.log(`□□Connected: ${peer.id}`);
      peer.send('Welcome!');
      peer.tags.add('authenticated');  // Tag for filtering
    },
    onMessage: (peer, message) => {
      console.log(`□□${message.text}`);
      peer.send(`Echo: ${message.text}`);
    },
    onClose: (peer, code, reason) => {
      console.log(`□□Disconnected: ${peer.id}`);
    },
    onError: (peer, error) => {
      console.error(`□ Error: ${error.message}`);
    },
  },
});
```

TypedRouter for Type-Safe RPC

Use `@api.global/typedrequest` for type-safe WebSocket communication:

```
import { TypedRouter } from '@api.global/typedrequest';

const typedRouter = new TypedRouter();
typedRouter.addTypedHandler(MyTypedRequest, async (request) => {
  return { result: 'processed' };
});

const server = new SmartServe({
  port: 3000,
```

```
  websocket: {
    typedRouter,  // Handles message routing automatically
    onConnectionOpen: (peer) => {
      peer.tags.add('subscriber');
    },
  },
});

// Broadcast to tagged connections
server.broadcast({ event: 'update' }, (peer) => peer.tags.has('subscriber'));
```

HTTPS/TLS

Enable HTTPS with certificate configuration:

```
import * as fs from 'fs';

const server = new SmartServe({
  port: 443,
  tls: {
    cert: fs.readFileSync('./cert.pem'),
    key: fs.readFileSync('./key.pem'),
    ca: fs.readFileSync('./ca.pem'),     // Optional: CA chain
    minVersion: 'TLSv1.2',               // Optional: minimum TLS version
    passphrase: 'optional-key-passphrase',
  },
});
```

Error Handling

Built-in HTTP error classes with factory methods:

```
import { HttpError, type IRequestContext } from '@push.rocks/smartserve';

@Route('/api')
```

```
class ApiController {
  @Get('/users/:id')
  async getUser(ctx: IRequestContext) {
    const user = await findUser(ctx.params.id);

    if (!user) {
      throw HttpError.notFound('User not found', { id: ctx.params.id });
    }

    return user;
  }
}

// Available factory methods:
HttpError.badRequest(message, details);      // 400
HttpError.unauthorized(message, details);    // 401
HttpError.forbidden(message, details);       // 403
HttpError.notFound(message, details);        // 404
HttpError.conflict(message, details);        // 409
HttpError.internal(message, details);        // 500
```

Global Error Handler

```
const server = new SmartServe({
  port: 3000,
  onError: (error, request) => {
    console.error('□□Server error:', error);

    // Return custom error response
    return new Response(
      JSON.stringify({ error: 'Something went wrong', requestId: crypto.randomUUID() }),
      { status: 500, headers: { 'Content-Type': 'application/json' } }
    );
  },
});
```

Request Context

Every handler receives a typed request context:

```
interface IRequestContext<TBody = unknown> {
  request: Request;              // Original Request (body never consumed by framework)
  params: Record<string, string>; // URL path parameters (/users/:id → { id: '123' })
  query: Record<string, string>; // Query string (?page=1 → { page: '1' })
  headers: Headers;              // Request headers
  path: string;                  // Matched route path
  method: THttpMethod;           // GET, POST, PUT, DELETE, etc.
  url: URL;                      // Full URL object
  runtime: 'node' | 'deno' | 'bun';
  state: Record<string, unknown>; // Per-request state (share data between interceptors)

  // □□Lazy body parsing (cached after first call)
  json(): Promise<TBody>;        // Parse as JSON (typed!)
  text(): Promise<string>;       // Parse as text
  arrayBuffer(): Promise<ArrayBuffer>;
  formData(): Promise<FormData>;
}
```

Lazy Body Parsing: The request body is only consumed when you call `json()`, `text()`, etc. This allows raw access to `ctx.request` for cases like webhook signature verification:

```
@Post('/webhook')
async handleWebhook(ctx: IRequestContext) {
  // Get raw body for signature verification
  const rawBody = await ctx.request.text();
  const signature = ctx.headers.get('X-Signature');

  if (!verifyHmac(rawBody, signature)) {
    throw HttpError.unauthorized('Invalid signature');
  }

  // Parse the body manually
  const payload = JSON.parse(rawBody);
  return { processed: true };
}
```

Custom Request Handler

Bypass decorator routing entirely for low-level control:

```
const server = new SmartServe({ port: 3000 });

server.setHandler(async (request, connectionInfo) => {
  const url = new URL(request.url);

  if (url.pathname === '/health') {
    return new Response('OK', { status: 200 });
  }

  if (url.pathname.startsWith('/api')) {
    // Handle API routes manually
    const body = await request.json();
    return new Response(JSON.stringify({ received: body }), {
      headers: { 'Content-Type': 'application/json' },
    });
  }

  return new Response('Not Found', { status: 404 });
});

await server.start();
```

Runtime Detection

SmartServe automatically detects and optimizes for the current runtime:

```
const instance = await server.start();

console.log(instance.runtime);   // 'node' | 'deno' | 'bun'
console.log(instance.port);      // 3000
console.log(instance.hostname);  // '0.0.0.0'
```

```
console.log(instance.secure);    // true if TLS enabled

// Server statistics
const stats = instance.stats();
console.log(stats.uptime);           // Seconds since start
console.log(stats.requestsTotal);    // Total requests handled
console.log(stats.requestsActive);   // Currently processing
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:13:43 UTC by foss.global Team

Updated 2026-03-28 12:20:29 UTC by foss.global Team