

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.11;
4
5 interface IBaseV1Factory {
6     function allPairsLength() external view returns
7         (uint);
8     function isPair(address pair) external view ret
9         urns (bool);
10    function pairCodeHash() external pure returns
11        (bytes32);
12    function getPair(address tokenA, address token,
13        bool stable) external view returns (address);
14    function createPair(address tokenA, address tok
15        enB, bool stable) external returns (address pair);
16 }
17
18 interface IBaseV1Pair {
19     function transferFrom(address src, address dst,
20         uint amount) external returns (bool);
21     function permit(address owner, address spender,
22         uint value, uint deadline, uint8 v, bytes32 r, byte
23         s32 s) external;
24     function swap(uint amount0Out, uint amount1Out,
25         address to, bytes calldata data) external;
26     function burn(address to) external returns (uin
27         t amount0, uint amount1);
28     function mint(address to) external returns (uin
29         t liquidity);
30     function getReserves() external view returns (u
31         int112 _reserve0, uint112 _reserve1, uint32 _blockT
32         imESTAMPLast);
33     function getAmountOut(uint, address) external v
34         iew returns (uint);
35 }
36
37 interface ERC20 {
38     function totalSupply() external view returns (u
39         int256);
40     function transfer(address recipient, uint amoun
41         t) external returns (bool);
42     function decimals() external view returns (uint
43         8);
44     function symbol() external view returns (string
45         memory);
46     function balanceOf(address) external view retur
47         ns (uint);
48     function transferFrom(address sender, address r
49         ecipient, uint amount) external returns (bool);
50     function approve(address spender, uint value) e
51         xternal returns (bool);
52 }
53
54 library Math {
55     function min(uint a, uint b) internal pure retu
56         rns (uint) {
57         return a < b ? a : b;
58     }
59     function sqrt(uint y) internal pure returns (ui
60         nt z) {
61         if (y > 3) {
62             z = y;
63             uint x = y / 2 + 1;

```

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.11;
4
5 interface IBaseV1Factory {
6     function allPairsLength() external view returns
7         (uint);
8     function isPair(address pair) external view ret
9         urns (bool);
10    function pairCodeHash() external pure returns
11        (bytes32);
12    function getPair(address tokenA, address token,
13        bool stable) external view returns (address);
14    function createPair(address tokenA, address tok
15        enB, bool stable) external returns (address pair);
16 }
17
18 interface IBaseV1Pair {
19     function transferFrom(address src, address dst,
20         uint amount) external returns (bool);
21     function permit(address owner, address spender,
22         uint value, uint deadline, uint8 v, bytes32 r, byte
23         s32 s) external;
24     function swap(uint amount0Out, uint amount1Out,
25         address to, bytes calldata data) external;
26     function burn(address to) external returns (uin
27         t amount0, uint amount1);
28     function mint(address to) external returns (uin
29         t liquidity);
30     function getReserves() external view returns (u
31         int112 _reserve0, uint112 _reserve1, uint32 _blockT
32         imESTAMPLast);
33     function getAmountOut(uint, address) external v
34         iew returns (uint);
35 }
36
37 interface ERC20 {
38     function totalSupply() external view returns (u
39         int256);
40     function transfer(address recipient, uint amoun
41         t) external returns (bool);
42     function decimals() external view returns (uint
43         8);
44     function symbol() external view returns (string
45         memory);
46     function balanceOf(address) external view retur
47         ns (uint);
48     function transferFrom(address sender, address r
49         ecipient, uint amount) external returns (bool);
50     function approve(address spender, uint value) e
51         xternal returns (bool);
52 }
53
54 library Math {
55     function min(uint a, uint b) internal pure retu
56         rns (uint) {
57         return a < b ? a : b;
58     }
59     function sqrt(uint y) internal pure returns (ui
60         nt z) {
61         if (y > 3) {
62             z = y;
63             uint x = y / 2 + 1;

```

```

41         while (x < z) {
42             z = x;
43             x = (y / x + x) / 2;
44         }
45     } else if (y != 0) {
46         z = 1;
47     }
48 }
49 }
50
51 interface IWFTM {
52     function deposit() external payable returns (uint);
53     function transfer(address to, uint value) external returns (bool);
54     function withdraw(uint) external returns (uint);
55 }
56
57 contract BaseV1Router01 {
58     struct route {
59         address from;
60         address to;
61         bool stable;
62     }
63
64     address public immutable factory;
65     IWFTM public immutable wftm;
66     uint internal constant MINIMUM_LIQUIDITY = 10**
67     3;
68     bytes32 immutable pairCodeHash;
69
70     modifier ensure(uint deadline) {
71         require(deadline >= block.timestamp, 'BaseV
1Router: EXPIRED');
72         _;
73     }
74
75     constructor(address _factory, address _wftm) {
76         factory = _factory;
77         pairCodeHash = IBaseV1Factory(_factory).pai
rCodeHash();
78         wftm = IWFTM(_wftm);
79     }
80
81     receive() external payable {
82         assert(msg.sender == address(wftm)); // onl
y accept ETH via fallback from the WETH contract
83     }
84
85     function sortTokens(address tokenA, address tok
enB) public pure returns (address token0, address t
oken1) {
86         require(tokenA != tokenB, 'BaseV1Router: ID
ENTICAL_ADDRESSES');
87         (token0, token1) = tokenA < tokenB ? (token
A, tokenB) : (tokenB, tokenA);
88         require(token0 != address(0), 'BaseV1Route
r: ZERO_ADDRESS');
89     }
90
91     // calculates the CREATE2 address for a pair wi
thout making any external calls
92     function pairFor(address tokenA, address token
B, bool stable) public view returns (address pair)
{
93         (address token0, address token1) = sortToke
ns(tokenA, tokenB);

```

```

41         while (x < z) {
42             z = x;
43             x = (y / x + x) / 2;
44         }
45     } else if (y != 0) {
46         z = 1;
47     }
48 }
49 }
50
51 interface IWFTM {
52     function deposit() external payable returns (ui
nt);
53     function transfer(address to, uint value) exter
nal returns (bool);
54     function withdraw(uint) external returns (uin
t);
55 }
56
57 contract BaseV1Router01 {
58     struct route {
59         address from;
60         address to;
61         bool stable;
62     }
63
64     address public immutable factory;
65     IWFTM public immutable wftm;
66     uint internal constant MINIMUM_LIQUIDITY = 10**
67     3;
68     bytes32 immutable pairCodeHash;
69
70     modifier ensure(uint deadline) {
71         require(deadline >= block.timestamp, 'BaseV
1Router: EXPIRED');
72         _;
73     }
74
75     constructor(address _factory, address _wftm) {
76         factory = _factory;
77         pairCodeHash = IBaseV1Factory(_factory).pai
rCodeHash();
78         wftm = IWFTM(_wftm);
79     }
80
81     receive() external payable {
82         assert(msg.sender == address(wftm)); // onl
y accept ETH via fallback from the WETH contract
83     }
84
85     function sortTokens(address tokenA, address tok
enB) public pure returns (address token0, address t
oken1) {
86         require(tokenA != tokenB, 'BaseV1Router: ID
ENTICAL_ADDRESSES');
87         (token0, token1) = tokenA < tokenB ? (token
A, tokenB) : (tokenB, tokenA);
88         require(token0 != address(0), 'BaseV1Route
r: ZERO_ADDRESS');
89     }
90
91     // calculates the CREATE2 address for a pair wi
thout making any external calls
92     function pairFor(address tokenA, address token
B, bool stable) public view returns (address pair)
{
93         (address token0, address token1) = sortToke
ns(tokenA, tokenB);

```

```

94     pair = address(uint160(uint256(keccak256(abi
i.encodePacked(
95         hex'ff',
96         factory,
97         keccak256(abi.encodePacked(token0, toke
n1, stable))),
98         pairCodeHash // init code hash
99         ))));
100 }
101
102 // given some amount of an asset and pair reser
ves, returns an equivalent amount of the other asse
t
103 function quoteLiquidity(uint amountA, uint rese
rveA, uint reserveB) internal pure returns (uint am
ountB) {
104     require(amountA > 0, 'BaseV1Router: INSUFFI
CIENT_AMOUNT');
105     require(reserveA > 0 && reserveB > 0, 'Base
V1Router: INSUFFICIENT_LIQUIDITY');
106     amountB = amountA * reserveB / reserveA;
107 }
108
109 // fetches and sorts the reserves for a pair
110 function getReserves(address tokenA, address to
kenB, bool stable) public view returns (uint reserv
eA, uint reserveB) {
111     (address token0,) = sortTokens(tokenA, toke
nB);
112     (uint reserve0, uint reserve1,) = IBaseV1Pa
ir(pairFor(tokenA, tokenB, stable)).getReserves();
113     (reserveA, reserveB) = tokenA == token0 ?
(reserve0, reserve1) : (reserve1, reserve0);
114 }
115
116 // performs chained getAmountOut calculations o
n any number of pairs
117 function getAmountOut(uint amountIn, address to
kenIn, address tokenOut) external view returns (uin
t amount, bool stable) {
118     address pair = pairFor(tokenIn, tokenOut, t
rue);
119     uint amountStable;
120     uint amountVolatile;
121     if (IBaseV1Factory(factory).isPair(pair)) {
122         amountStable = IBaseV1Pair(pair).getAmo
untOut(amountIn, tokenIn);
123     }
124     pair = pairFor(tokenIn, tokenOut, false);
125     if (IBaseV1Factory(factory).isPair(pair)) {
126         amountVolatile = IBaseV1Pair(pair).getA
mountOut(amountIn, tokenIn);
127     }
128     return amountStable > amountVolatile ? (amo
untStable, true) : (amountVolatile, false);
129 }
130
131 // performs chained getAmountOut calculations o
n any number of pairs
132 function getAmountsOut(uint amountIn, route[] m
emory routes) public view returns (uint[] memory am
ounts) {
133     require(routes.length >= 1, 'BaseV1Router:
INVALID_PATH');
134     amounts = new uint[](routes.length+1);
135     amounts[0] = amountIn;
136     for (uint i = 0; i < routes.length; i++) {

```

```

94     pair = address(uint160(uint256(keccak256(abi
i.encodePacked(
95         hex'ff',
96         factory,
97         keccak256(abi.encodePacked(token0, toke
n1, stable))),
98         pairCodeHash // init code hash
99         ))));
100 }
101
102 // given some amount of an asset and pair reser
ves, returns an equivalent amount of the other asse
t
103 function quoteLiquidity(uint amountA, uint rese
rveA, uint reserveB) internal pure returns (uint am
ountB) {
104     require(amountA > 0, 'BaseV1Router: INSUFFI
CIENT_AMOUNT');
105     require(reserveA > 0 && reserveB > 0, 'Base
V1Router: INSUFFICIENT_LIQUIDITY');
106     amountB = amountA * reserveB / reserveA;
107 }
108
109 // fetches and sorts the reserves for a pair
110 function getReserves(address tokenA, address to
kenB, bool stable) public view returns (uint reserv
eA, uint reserveB) {
111     (address token0,) = sortTokens(tokenA, toke
nB);
112     (uint reserve0, uint reserve1,) = IBaseV1Pa
ir(pairFor(tokenA, tokenB, stable)).getReserves();
113     (reserveA, reserveB) = tokenA == token0 ?
(reserve0, reserve1) : (reserve1, reserve0);
114 }
115
116 // performs chained getAmountOut calculations o
n any number of pairs
117 function getAmountOut(uint amountIn, address to
kenIn, address tokenOut) external view returns (uin
t amount, bool stable) {
118     address pair = pairFor(tokenIn, tokenOut, t
rue);
119     uint amountStable;
120     uint amountVolatile;
121     if (IBaseV1Factory(factory).isPair(pair)) {
122         amountStable = IBaseV1Pair(pair).getAmo
untOut(amountIn, tokenIn);
123     }
124     pair = pairFor(tokenIn, tokenOut, false);
125     if (IBaseV1Factory(factory).isPair(pair)) {
126         amountVolatile = IBaseV1Pair(pair).getA
mountOut(amountIn, tokenIn);
127     }
128     return amountStable > amountVolatile ? (amo
untStable, true) : (amountVolatile, false);
129 }
130
131 // performs chained getAmountOut calculations o
n any number of pairs
132 function getAmountsOut(uint amountIn, route[] m
emory routes) public view returns (uint[] memory am
ounts) {
133     require(routes.length >= 1, 'BaseV1Router:
INVALID_PATH');
134     amounts = new uint[](routes.length+1);
135     amounts[0] = amountIn;
136     for (uint i = 0; i < routes.length; i++) {

```

```

137         address pair = pairFor(routes[i].from,
138             routes[i].to, routes[i].stable);
139         if (IBaseV1Factory(factory).isPair(pair)) {
140             amounts[i+1] = IBaseV1Pair(pair).getAmountOut(amounts[i], routes[i].from);
141         }
142     }
143
144     function isPair(address pair) external view returns (bool) {
145         return IBaseV1Factory(factory).isPair(pair);
146     }
147
148     function quoteAddLiquidity(
149         address tokenA,
150         address tokenB,
151         bool stable,
152         uint amountADesired,
153         uint amountBDesired
154     ) external view returns (uint amountA, uint amountB, uint liquidity) {
155         // create the pair if it doesn't exist yet
156         address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);
157         (uint reserveA, uint reserveB) = (0,0);
158         uint _totalSupply = 0;
159         if (_pair != address(0)) {
160             _totalSupply = erc20(_pair).totalSupply();
161             (reserveA, reserveB) = getReserves(tokenA, tokenB, stable);
162         }
163         if (reserveA == 0 && reserveB == 0) {
164             (amountA, amountB) = (amountADesired, amountBDesired);
165             liquidity = Math.sqrt(amountA * amountB) - MINIMUM_LIQUIDITY;
166         } else {
167             uint amountBOptimal = quoteLiquidity(amountADesired, reserveA, reserveB);
168             if (amountBOptimal <= amountBDesired) {
169                 (amountA, amountB) = (amountADesired, amountBOptimal);
170                 liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
171             } else {
172                 uint amountAOptimal = quoteLiquidity(amountBDesired, reserveB, reserveA);
173                 (amountA, amountB) = (amountAOptimal, amountBDesired);
174                 liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
175             }
176         }
177     }
178 }
179
180 function quoteRemoveLiquidity(
181     address tokenA,
182     address tokenB,
183     bool stable,
184     uint liquidity

```

```

137         address pair = pairFor(routes[i].from,
138             routes[i].to, routes[i].stable);
139         if (IBaseV1Factory(factory).isPair(pair)) {
140             amounts[i+1] = IBaseV1Pair(pair).getAmountOut(amounts[i], routes[i].from);
141         }
142     }
143
144     function isPair(address pair) external view returns (bool) {
145         return IBaseV1Factory(factory).isPair(pair);
146     }
147
148     function quoteAddLiquidity(
149         address tokenA,
150         address tokenB,
151         bool stable,
152         uint amountADesired,
153         uint amountBDesired
154     ) external view returns (uint amountA, uint amountB, uint liquidity) {
155         // create the pair if it doesn't exist yet
156         address _pair = IBaseV1Factory(factory).getPair(tokenA, tokenB, stable);
157         (uint reserveA, uint reserveB) = (0,0);
158         uint _totalSupply = 0;
159         if (_pair != address(0)) {
160             _totalSupply = erc20(_pair).totalSupply();
161             (reserveA, reserveB) = getReserves(tokenA, tokenB, stable);
162         }
163         if (reserveA == 0 && reserveB == 0) {
164             (amountA, amountB) = (amountADesired, amountBDesired);
165             liquidity = Math.sqrt(amountA * amountB) - MINIMUM_LIQUIDITY;
166         } else {
167             uint amountBOptimal = quoteLiquidity(amountADesired, reserveA, reserveB);
168             if (amountBOptimal <= amountBDesired) {
169                 (amountA, amountB) = (amountADesired, amountBOptimal);
170                 liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
171             } else {
172                 uint amountAOptimal = quoteLiquidity(amountBDesired, reserveB, reserveA);
173                 (amountA, amountB) = (amountAOptimal, amountBDesired);
174                 liquidity = Math.min(amountA * _totalSupply / reserveA, amountB * _totalSupply / reserveB);
175             }
176         }
177     }
178 }
179
180 function quoteRemoveLiquidity(
181     address tokenA,
182     address tokenB,
183     bool stable,
184     uint liquidity

```

```

185     ) external view returns (uint amountA, uint amo
untB) {
186         // create the pair if it doesn't exist yet
187         address _pair = IBaseV1Factory(factory).get
Pair(tokenA, tokenB, stable);
188
189         if (_pair == address(0)) {
190             return (0,0);
191         }
192
193         (uint reserveA, uint reserveB) = getReserve
s(tokenA, tokenB, stable);
194         uint _totalSupply = erc20(_pair).totalSuppl
y();
195
196         amountA = liquidity * reserveA / _totalSupp
ly; // using balances ensures pro-rata distribution
197         amountB = liquidity * reserveB / _totalSupp
ly; // using balances ensures pro-rata distribution
198
199     }
200
201     function _addLiquidity(
202         address tokenA,
203         address tokenB,
204         bool stable,
205         uint amountADesired,
206         uint amountBDesired,
207         uint amountAMin,
208         uint amountBMin
209     ) internal returns (uint amountA, uint amountB)
{
210         require(amountADesired >= amountAMin);
211         require(amountBDesired >= amountBMin);
212         // create the pair if it doesn't exist yet
213         address _pair = IBaseV1Factory(factory).get
Pair(tokenA, tokenB, stable);
214         if (_pair == address(0)) {
215             _pair = IBaseV1Factory(factory).createP
air(tokenA, tokenB, stable);
216         }
217         (uint reserveA, uint reserveB) = getReserve
s(tokenA, tokenB, stable);
218         if (reserveA == 0 && reserveB == 0) {
219             (amountA, amountB) = (amountADesired, a
mountBDesired);
220         } else {
221             uint amountBOptimal = quoteLiquidity(am
ountADesired, reserveA, reserveB);
222             if (amountBOptimal <= amountBDesired) {
223                 require(amountBOptimal >= amountBMin, 'BaseV1Router: INSUFFICIENT_B_AMOUNT');
224                 (amountA, amountB) = (amountADesire
d, amountBOptimal);
225             } else {
226                 uint amountAOptimal = quoteLiquidit
y(amountBDesired, reserveB, reserveA);
227                 assert(amountAOptimal <= amountADes
ired);
228                 require(amountAOptimal >= amountAMi
n, 'BaseV1Router: INSUFFICIENT_A_AMOUNT');
229                 (amountA, amountB) = (amountAOptima
l, amountBDesired);
230             }
231         }
232     }
233
234     function addLiquidity(
235         address tokenA,

```

```

185     ) external view returns (uint amountA, uint amo
untB) {
186         // create the pair if it doesn't exist yet
187         address _pair = IBaseV1Factory(factory).get
Pair(tokenA, tokenB, stable);
188
189         if (_pair == address(0)) {
190             return (0,0);
191         }
192
193         (uint reserveA, uint reserveB) = getReserve
s(tokenA, tokenB, stable);
194         uint _totalSupply = erc20(_pair).totalSuppl
y();
195
196         amountA = liquidity * reserveA / _totalSupp
ly; // using balances ensures pro-rata distribution
197         amountB = liquidity * reserveB / _totalSupp
ly; // using balances ensures pro-rata distribution
198
199     }
200
201     function _addLiquidity(
202         address tokenA,
203         address tokenB,
204         bool stable,
205         uint amountADesired,
206         uint amountBDesired,
207         uint amountAMin,
208         uint amountBMin
209     ) internal returns (uint amountA, uint amountB)
{
210         require(amountADesired >= amountAMin);
211         require(amountBDesired >= amountBMin);
212         // create the pair if it doesn't exist yet
213         address _pair = IBaseV1Factory(factory).get
Pair(tokenA, tokenB, stable);
214         if (_pair == address(0)) {
215             _pair = IBaseV1Factory(factory).createP
air(tokenA, tokenB, stable);
216         }
217         (uint reserveA, uint reserveB) = getReserve
s(tokenA, tokenB, stable);
218         if (reserveA == 0 && reserveB == 0) {
219             (amountA, amountB) = (amountADesired, a
mountBDesired);
220         } else {
221             uint amountBOptimal = quoteLiquidity(am
ountADesired, reserveA, reserveB);
222             if (amountBOptimal <= amountBDesired) {
223                 require(amountBOptimal >= amountBMin, 'BaseV1Router: INSUFFICIENT_B_AMOUNT');
224                 (amountA, amountB) = (amountADesire
d, amountBOptimal);
225             } else {
226                 uint amountAOptimal = quoteLiquidit
y(amountBDesired, reserveB, reserveA);
227                 assert(amountAOptimal <= amountADes
ired);
228                 require(amountAOptimal >= amountAMi
n, 'BaseV1Router: INSUFFICIENT_A_AMOUNT');
229                 (amountA, amountB) = (amountAOptima
l, amountBDesired);
230             }
231         }
232     }
233
234     function addLiquidity(
235         address tokenA,

```

```

236     address tokenB,
237     bool stable,
238     uint amountADesired,
239     uint amountBDesired,
240     uint amountAMin,
241     uint amountBMin,
242     address to,
243     uint deadline
244     ) external ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
245         (amountA, amountB) = _addLiquidity(tokenA,
246         tokenB, stable, amountADesired, amountBDesired, amountAMin, amountBMin);
247         address pair = pairFor(tokenA, tokenB, stable);
248         _safeTransferFrom(tokenA, msg.sender, pair, amountA);
249         _safeTransferFrom(tokenB, msg.sender, pair, amountB);
250         liquidity = IBaseV1Pair(pair).mint(to);
251     }
252     function addLiquidityFTM(
253         address token,
254         bool stable,
255         uint amountTokenDesired,
256         uint amountTokenMin,
257         uint amountFTMMin,
258         address to,
259         uint deadline
260         ) external payable ensure(deadline) returns (uint amountToken, uint amountFTM, uint liquidity) {
261         (amountToken, amountFTM) = _addLiquidity(
262             token,
263             address(wftm),
264             stable,
265             amountTokenDesired,
266             msg.value,
267             amountTokenMin,
268             amountFTMMin
269         );
270         address pair = pairFor(token, address(wftm), stable);
271         _safeTransferFrom(token, msg.sender, pair, amountToken);
272         wftm.deposit{value: amountFTM}();
273         assert(wftm.transfer(pair, amountFTM));
274         liquidity = IBaseV1Pair(pair).mint(to);
275         // refund dust eth, if any
276         if (msg.value > amountFTM) _safeTransferFTM(msg.sender, msg.value - amountFTM);
277     }
278
279     // **** REMOVE LIQUIDITY ****
280     function removeLiquidity(
281         address tokenA,
282         address tokenB,
283         bool stable,
284         uint liquidity,
285         uint amountAMin,
286         uint amountBMin,
287         address to,
288         uint deadline
289         ) public ensure(deadline) returns (uint amountA, uint amountB) {
290         address pair = pairFor(tokenA, tokenB, stable);

```

```

236     address tokenB,
237     bool stable,
238     uint amountADesired,
239     uint amountBDesired,
240     uint amountAMin,
241     uint amountBMin,
242     address to,
243     uint deadline
244     ) external ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
245         (amountA, amountB) = _addLiquidity(tokenA,
246         tokenB, stable, amountADesired, amountBDesired, amountAMin, amountBMin);
247         address pair = pairFor(tokenA, tokenB, stable);
248         _safeTransferFrom(tokenA, msg.sender, pair, amountA);
249         _safeTransferFrom(tokenB, msg.sender, pair, amountB);
250         liquidity = IBaseV1Pair(pair).mint(to);
251     }
252     function addLiquidityFTM(
253         address token,
254         bool stable,
255         uint amountTokenDesired,
256         uint amountTokenMin,
257         uint amountFTMMin,
258         address to,
259         uint deadline
260         ) external payable ensure(deadline) returns (uint amountToken, uint amountFTM, uint liquidity) {
261         (amountToken, amountFTM) = _addLiquidity(
262             token,
263             address(wftm),
264             stable,
265             amountTokenDesired,
266             msg.value,
267             amountTokenMin,
268             amountFTMMin
269         );
270         address pair = pairFor(token, address(wftm), stable);
271         _safeTransferFrom(token, msg.sender, pair, amountToken);
272         wftm.deposit{value: amountFTM}();
273         assert(wftm.transfer(pair, amountFTM));
274         liquidity = IBaseV1Pair(pair).mint(to);
275         // refund dust eth, if any
276         if (msg.value > amountFTM) _safeTransferFTM(msg.sender, msg.value - amountFTM);
277     }
278
279     // **** REMOVE LIQUIDITY ****
280     function removeLiquidity(
281         address tokenA,
282         address tokenB,
283         bool stable,
284         uint liquidity,
285         uint amountAMin,
286         uint amountBMin,
287         address to,
288         uint deadline
289         ) public ensure(deadline) returns (uint amountA, uint amountB) {
290         address pair = pairFor(tokenA, tokenB, stable);

```

```

291     require(IBaseV1Pair(pair).transferFrom(msg.
sender, pair, liquidity)); // send liquidity to pai
r
292     (uint amount0, uint amount1) = IBaseV1Pair
(pair).burn(to);
293     (address token0,) = sortTokens(tokenA, toke
nB);
294     (amountA, amountB) = tokenA == token0 ? (am
ount0, amount1) : (amount1, amount0);
295     require(amountA >= amountAMin, 'BaseV1Route
r: INSUFFICIENT_A_AMOUNT');
296     require(amountB >= amountBMin, 'BaseV1Route
r: INSUFFICIENT_B_AMOUNT');
297 }
298
299 function removeLiquidityFTM(
300     address token,
301     bool stable,
302     uint liquidity,
303     uint amountTokenMin,
304     uint amountFTMMin,
305     address to,
306     uint deadline
307 ) public ensure(deadline) returns (uint amountT
oken, uint amountFTM) {
308     (amountToken, amountFTM) = removeLiquidity(
309         token,
310         address(wftm),
311         stable,
312         liquidity,
313         amountTokenMin,
314         amountFTMMin,
315         address(this),
316         deadline
317     );
318     _safeTransfer(token, to, amountToken);
319     wftm.withdraw(amountFTM);
320     _safeTransferFTM(to, amountFTM);
321 }
322
323 function removeLiquidityWithPermit(
324     address tokenA,
325     address tokenB,
326     bool stable,
327     uint liquidity,
328     uint amountAMin,
329     uint amountBMin,
330     address to,
331     uint deadline,
332     bool approveMax, uint8 v, bytes32 r, bytes3
2 s
333 ) external returns (uint amountA, uint amountB)
{
334     address pair = pairFor(tokenA, tokenB, stab
le);
335     {
336         uint value = approveMax ? type(uint).ma
x : liquidity;
337         IBaseV1Pair(pair).permit(msg.sender, ad
dress(this), value, deadline, v, r, s);
338     }
339
340     (amountA, amountB) = removeLiquidity(token
A, tokenB, stable, liquidity, amountAMin, amountBMin, to, deadline);
341 }
342
343 function removeLiquidityFTMWithPermit(

```

```

291     require(IBaseV1Pair(pair).transferFrom(msg.
sender, pair, liquidity)); // send liquidity to pai
r
292     (uint amount0, uint amount1) = IBaseV1Pair
(pair).burn(to);
293     (address token0,) = sortTokens(tokenA, toke
nB);
294     (amountA, amountB) = tokenA == token0 ? (am
ount0, amount1) : (amount1, amount0);
295     require(amountA >= amountAMin, 'BaseV1Route
r: INSUFFICIENT_A_AMOUNT');
296     require(amountB >= amountBMin, 'BaseV1Route
r: INSUFFICIENT_B_AMOUNT');
297 }
298
299 function removeLiquidityFTM(
300     address token,
301     bool stable,
302     uint liquidity,
303     uint amountTokenMin,
304     uint amountFTMMin,
305     address to,
306     uint deadline
307 ) public ensure(deadline) returns (uint amountT
oken, uint amountFTM) {
308     (amountToken, amountFTM) = removeLiquidity(
309         token,
310         address(wftm),
311         stable,
312         liquidity,
313         amountTokenMin,
314         amountFTMMin,
315         address(this),
316         deadline
317     );
318     _safeTransfer(token, to, amountToken);
319     wftm.withdraw(amountFTM);
320     _safeTransferFTM(to, amountFTM);
321 }
322
323 function removeLiquidityWithPermit(
324     address tokenA,
325     address tokenB,
326     bool stable,
327     uint liquidity,
328     uint amountAMin,
329     uint amountBMin,
330     address to,
331     uint deadline,
332     bool approveMax, uint8 v, bytes32 r, bytes3
2 s
333 ) external returns (uint amountA, uint amountB)
{
334     address pair = pairFor(tokenA, tokenB, stab
le);
335     {
336         uint value = approveMax ? type(uint).ma
x : liquidity;
337         IBaseV1Pair(pair).permit(msg.sender, ad
dress(this), value, deadline, v, r, s);
338     }
339
340     (amountA, amountB) = removeLiquidity(token
A, tokenB, stable, liquidity, amountAMin, amountBMin, to, deadline);
341 }
342
343 function removeLiquidityFTMWithPermit(

```



```

344     address token,
345     bool stable,
346     uint liquidity,
347     uint amountTokenMin,
348     uint amountFTMMin,
349     address to,
350     uint deadline,
351     bool approveMax, uint8 v, bytes32 r, bytes3
2 s
352     ) external returns (uint amountToken, uint amou
ntFTM) {
353         address pair = pairFor(token, address(wft
m), stable);
354         uint value = approveMax ? type(uint).max :
liquidity;
355         IBaseV1Pair(pair).permit(msg.sender, addres
s(this), value, deadline, v, r, s);
356         (amountToken, amountFTM) = removeLiquidityF
TM(token, stable, liquidity, amountTokenMin, amount
FTMMin, to, deadline);
357     }
358
359     // **** SWAP ****
360     // requires the initial amount to have already
been sent to the first pair
361     function _swap(uint[] memory amounts, route[] m
emory routes, address _to) internal virtual {
362         for (uint i = 0; i < routes.length; i++) {
363             (address token0,) = sortTokens(routes
[i].from, routes[i].to);
364             uint amountOut = amounts[i + 1];
365             (uint amount0Out, uint amount1Out) = ro
utes[i].from == token0 ? (uint(0), amountOut) : (am
ountOut, uint(0));
366             address to = i < routes.length - 1 ? pa
irFor(routes[i+1].from, routes[i+1].to, routes[i+
1].stable) : _to;
367             IBaseV1Pair(pairFor(routes[i].from, rou
tes[i].to, routes[i].stable)).swap(
368                 amount0Out, amount1Out, to, new byt
es(0)
369             );
370         }
371     }
372
373     function swapExactTokensForTokensSimple(
374         uint amountIn,
375         uint amountOutMin,
376         address tokenFrom,
377         address tokenTo,
378         bool stable,
379         address to,
380         uint deadline
381     ) external ensure(deadline) returns (uint[] mem
ory amounts) {
382         route[] memory routes = new route[](1);
383         routes[0].from = tokenFrom;
384         routes[0].to = tokenTo;
385         routes[0].stable = stable;
386         amounts = getAmountsOut(amountIn, routes);
387         require(amounts[amounts.length - 1] >= amou
ntOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUN
T');
388         _safeTransferFrom(
389             routes[0].from, msg.sender, pairFor(rou
tes[0].from, routes[0].to, routes[0].stable), amoun
ts[0]
390         );

```

```

344     address token,
345     bool stable,
346     uint liquidity,
347     uint amountTokenMin,
348     uint amountFTMMin,
349     address to,
350     uint deadline,
351     bool approveMax, uint8 v, bytes32 r, bytes3
2 s
352     ) external returns (uint amountToken, uint amou
ntFTM) {
353         address pair = pairFor(token, address(wft
m), stable);
354         uint value = approveMax ? type(uint).max :
liquidity;
355         IBaseV1Pair(pair).permit(msg.sender, addres
s(this), value, deadline, v, r, s);
356         (amountToken, amountFTM) = removeLiquidityF
TM(token, stable, liquidity, amountTokenMin, amount
FTMMin, to, deadline);
357     }
358
359     // **** SWAP ****
360     // requires the initial amount to have already
been sent to the first pair
361     function _swap(uint[] memory amounts, route[] m
emory routes, address _to) internal virtual {
362         for (uint i = 0; i < routes.length; i++) {
363             (address token0,) = sortTokens(routes
[i].from, routes[i].to);
364             uint amountOut = amounts[i + 1];
365             (uint amount0Out, uint amount1Out) = ro
utes[i].from == token0 ? (uint(0), amountOut) : (am
ountOut, uint(0));
366             address to = i < routes.length - 1 ? pa
irFor(routes[i+1].from, routes[i+1].to, routes[i+
1].stable) : _to;
367             IBaseV1Pair(pairFor(routes[i].from, rou
tes[i].to, routes[i].stable)).swap(
368                 amount0Out, amount1Out, to, new byt
es(0)
369             );
370         }
371     }
372
373     function swapExactTokensForTokensSimple(
374         uint amountIn,
375         uint amountOutMin,
376         address tokenFrom,
377         address tokenTo,
378         bool stable,
379         address to,
380         uint deadline
381     ) external ensure(deadline) returns (uint[] mem
ory amounts) {
382         route[] memory routes = new route[](1);
383         routes[0].from = tokenFrom;
384         routes[0].to = tokenTo;
385         routes[0].stable = stable;
386         amounts = getAmountsOut(amountIn, routes);
387         require(amounts[amounts.length - 1] >= amou
ntOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUN
T');
388         _safeTransferFrom(
389             routes[0].from, msg.sender, pairFor(rou
tes[0].from, routes[0].to, routes[0].stable), amoun
ts[0]
390         );

```



```

391     _swap(amounts, routes, to);
392 }
393
394 function swapExactTokensForTokens(
395     uint amountIn,
396     uint amountOutMin,
397     route[] calldata routes,
398     address to,
399     uint deadline
400 ) external ensure(deadline) returns (uint[] memory amounts) {
401     amounts = getAmountsOut(amountIn, routes);
402     require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
403     _safeTransferFrom(
404         routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
405     );
406     _swap(amounts, routes, to);
407 }
408
409 function swapExactFTMForTokens(uint amountOutMin, route[] calldata routes, address to, uint deadline)
410     external
411     payable
412     ensure(deadline)
413     returns (uint[] memory amounts)
414     {
415         require(routes[0].from == address(wftm), 'BaseV1Router: INVALID_PATH');
416         amounts = getAmountsOut(msg.value, routes);
417         require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
418         wftm.deposit{value: amounts[0]}();
419         assert(wftm.transfer(pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]));
420         _swap(amounts, routes, to);
421     }
422
423 function swapExactTokensForFTM(uint amountIn, uint amountOutMin, route[] calldata routes, address to, uint deadline)
424     external
425     ensure(deadline)
426     returns (uint[] memory amounts)
427     {
428         require(routes[routes.length - 1].to == address(wftm), 'BaseV1Router: INVALID_PATH');
429         amounts = getAmountsOut(amountIn, routes);
430         require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
431         _safeTransferFrom(
432             routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
433         );
434         _swap(amounts, routes, address(this));
435         wftm.withdraw(amounts[amounts.length - 1]);
436         _safeTransferFTM(to, amounts[amounts.length - 1]);
437     }
438
439 function UNSAFE_swapExactTokensForTokens(

```

```

391     _swap(amounts, routes, to);
392 }
393
394 function swapExactTokensForTokens(
395     uint amountIn,
396     uint amountOutMin,
397     route[] calldata routes,
398     address to,
399     uint deadline
400 ) external ensure(deadline) returns (uint[] memory amounts) {
401     amounts = getAmountsOut(amountIn, routes);
402     require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
403     _safeTransferFrom(
404         routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
405     );
406     _swap(amounts, routes, to);
407 }
408
409 function swapExactFTMForTokens(uint amountOutMin, route[] calldata routes, address to, uint deadline)
410     external
411     payable
412     ensure(deadline)
413     returns (uint[] memory amounts)
414     {
415         require(routes[0].from == address(wftm), 'BaseV1Router: INVALID_PATH');
416         amounts = getAmountsOut(msg.value, routes);
417         require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
418         wftm.deposit{value: amounts[0]}();
419         assert(wftm.transfer(pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]));
420         _swap(amounts, routes, to);
421     }
422
423 function swapExactTokensForFTM(uint amountIn, uint amountOutMin, route[] calldata routes, address to, uint deadline)
424     external
425     ensure(deadline)
426     returns (uint[] memory amounts)
427     {
428         require(routes[routes.length - 1].to == address(wftm), 'BaseV1Router: INVALID_PATH');
429         amounts = getAmountsOut(amountIn, routes);
430         require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router: INSUFFICIENT_OUTPUT_AMOUNT');
431         _safeTransferFrom(
432             routes[0].from, msg.sender, pairFor(routes[0].from, routes[0].to, routes[0].stable), amounts[0]
433         );
434         _swap(amounts, routes, address(this));
435         wftm.withdraw(amounts[amounts.length - 1]);
436         _safeTransferFTM(to, amounts[amounts.length - 1]);
437     }
438
439 function UNSAFE_swapExactTokensForTokens(

```

```

440     uint[] memory amounts,
441     route[] calldata routes,
442     address to,
443     uint deadline
444 ) external ensure(deadline) returns (uint[] mem
ory) {
445     _safeTransferFrom(routes[0].from, msg.sende
r, pairFor(routes[0].from, routes[0].to, routes[0].
stable), amounts[0]);
446     _swap(amounts, routes, to);
447     return amounts;
448 }
449
450 function _safeTransferFTM(address to, uint valu
e) internal {
451     (bool success,) = to.call{value:value}(new
bytes(0));
452     require(success, 'TransferHelper: ETH_TRANS
FER_FAILED');
453 }
454
455 function _safeTransfer(address token, address t
o, uint256 value) internal {
456     require(token.code.length > 0);
457     (bool success, bytes memory data) =
458     token.call(abi.encodeWithSelector(erc20.tra
nsfer.selector, to, value));
459     require(success && (data.length == 0 || ab
i.decode(data, (bool))));
460 }
461
462 function _safeTransferFrom(address token, addre
ss from, address to, uint256 value) internal {
463     require(token.code.length > 0);
464     (bool success, bytes memory data) =
465     token.call(abi.encodeWithSelector(erc20.tra
nsferFrom.selector, from, to, value));
466     require(success && (data.length == 0 || ab
i.decode(data, (bool))));
467 }
468 }
469

```

```

440     uint[] memory amounts,
441     route[] calldata routes,
442     address to,
443     uint deadline
444 ) external ensure(deadline) returns (uint[] mem
ory) {
445     _safeTransferFrom(routes[0].from, msg.sende
r, pairFor(routes[0].from, routes[0].to, routes[0].
stable), amounts[0]);
446     _swap(amounts, routes, to);
447     return amounts;
448 }
449
450 function _safeTransferFTM(address to, uint valu
e) internal {
451     (bool success,) = to.call{value:value}(new
bytes(0));
452     require(success, 'TransferHelper: ETH_TRANS
FER_FAILED');
453 }
454
455 function _safeTransfer(address token, address t
o, uint256 value) internal {
456     require(token.code.length > 0);
457     (bool success, bytes memory data) =
458     token.call(abi.encodeWithSelector(erc20.tra
nsfer.selector, to, value));
459     require(success && (data.length == 0 || ab
i.decode(data, (bool))));
460 }
461
462 function _safeTransferFrom(address token, addre
ss from, address to, uint256 value) internal {
463     require(token.code.length > 0);
464     (bool success, bytes memory data) =
465     token.call(abi.encodeWithSelector(erc20.tra
nsferFrom.selector, from, to, value));
466     require(success && (data.length == 0 || ab
i.decode(data, (bool))));
467 }
468 }

```