In [1]:
```
import duckdb
import plotly.io as pio
from IPython.display import display

con = duckdb.execute("""IMPORT DATABASE '../data/db';""")
df = con.execute("""
                 SELECT *
                 FROM tasks
                 """).df()
con.close()
df
```

Out[1]:

In [2]:
```
import pandas as pd

ships = set(df["ship_id"])
roles = set(df["role"])


def _sort_tasks(task_list):
    return sorted(task_list, key=lambda t: pd.to_datetime(t.get("start_ts")))


tasks = {}
for ship in ships:
    tasks[ship] = {}

for _, row in df.iterrows():
    ship = row["ship_id"]
    role = row["role"]
    task_list = row["tasks"]
    tasks[ship][role] = _sort_tasks(task_list)
```

In [3]:
```
import plotly.graph_objects as go


# Build prepared data per (ship, role)
def prepare_tasks_for(ship_id, role_name):
    task_list = tasks.get(ship_id, {}).get(role_name, [])
    prepared = []
```

```
    for t in task_list:
        start = pd.to_datetime(t.get("start_ts"))
        end = pd.to_datetime(t.get("end_ts"))
        if pd.isna(start) or pd.isna(end) or end <= start:
            continue
        prepared.append({
            "start": start,
            "end": end,
            "task_id": t.get("task_id") or t.get("id") or None,
            "sailing_id": t.get("sailing_id")
        })
    if not prepared:
        return [], [], [], [], [], ship_id, role_name

    prepared.sort(key=lambda x: x["start"])

    # Greedy lane assignment to avoid overlaps
    lanes_end = []
    lane_idx_for_task = []
    for task in prepared:
        placed = False
        for i, lane_end in enumerate(lanes_end):
            if task["start"] >= lane_end:
                lane_idx_for_task.append(i)
                lanes_end[i] = task["end"]
                placed = True
                break
        if not placed:
            lane_idx_for_task.append(len(lanes_end))
            lanes_end.append(task["end"])

    t0 = prepared[0]["start"]
    start_hours = [(t["start"] - t0).total_seconds() / 3600 for t in prepare
    end_hours = [(t["end"] - t0).total_seconds() / 3600 for t in prepared]
    dur_hours = [eh - sh for sh, eh in zip(start_hours, end_hours)]
    y_vals = [i + 1 for i in lane_idx_for_task]
    return prepared, start_hours, dur_hours, y_vals, t0, ship_id, role_name


# Collect all combinations
ship_role_pairs = []
for s in sorted(tasks.keys()):
    for r in sorted(tasks[s].keys()):
        ship_role_pairs.append((s, r))

# Precompute traces for each combination
combo_traces = []
combo_meta = []  # metadata for each combo: (ship, role, n_traces, title, he
for (s, r) in ship_role_pairs:
    prepared, start_hours, dur_hours, y_vals, t0, ship_id, role_name = prepa
    traces = []
    for idx, (sh, dh, y, t) in enumerate(zip(start_hours, dur_hours, y_vals,
        label = t["task_id"] if t["task_id"] is not None else f"{idx + 1}"
        sailing_label = t.get("sailing_id") or "N/A"
        hovertext = (
            f"Ship: {ship_id}<br>Role: {role_name}"
```

```
            f"<br>Task ID: {label}"
            f"<br>Sailing ID: {sailing_label}"
            f"<br>Start: {t['start']}"
            f"<br>End: {t['end']}"
            f"<br>Duration (h): {dh:.2f}"
        )
        traces.append(go.Bar(
            x=[dh],
            y=[y],
            base=[sh],
            orientation="h",
            name=f"Task {label} | Sailing {sailing_label}",
            hovertext=hovertext,
            hoverinfo="text",
            marker=dict(line=dict(width=0.5, color="#777")),
            showlegend=True
        ))
    max_lane = max(y_vals) if y_vals else 1
    title = f"Tasks for Ship {ship_id} - Role {role_name}"
    height = max(420, 140 + 40 * max_lane)
    combo_traces.append(traces)
    combo_meta.append((s, r, len(traces), title, height))

# Build a single figure with all traces, only first combo visible
fig = go.Figure()
visibility = []

for ci, traces in enumerate(combo_traces):
    for tr in traces:
        fig.add_trace(tr)
        visibility.append(ci == 0)  # only first combo visible

# Update layout for initial combo with dark theme
_, _, _, init_title, init_height = combo_meta[0]
fig.update_layout(
    template="plotly_dark",
    title=init_title,
    xaxis_title="Time (hours since first task start)",
    yaxis_title="Overlap lane",
    bargap=0.2,
    barmode="overlay",
    height=init_height,
    legend_title_text="Tasks (ID | sailing_id)",
    paper_bgcolor="#111111",
    plot_bgcolor="#111111",
    font=dict(color="#e6e6e6")
)
fig.update_yaxes(dtick=1, gridcolor="#333333")
fig.update_xaxes(gridcolor="#333333")

# Build dropdown for filtering (combined Ship & Role)
buttons = []
trace_offset = 0
combo_offsets = []
for (_, _, ntr, _, _) in combo_meta:
    combo_offsets.append((trace_offset, trace_offset + ntr))
```

```
    trace_offset += ntr

for ci, (s, r, ntr, title, height) in enumerate(combo_meta):
    vis = [False] * len(visibility)
    start_idx, end_idx = combo_offsets[ci]
    for i in range(start_idx, end_idx):
        vis[i] = True
    buttons.append(dict(
        label=f"{s} | {r}",
        method="update",
        args=[
            {"visible": vis},
            {"title": title, "height": height}
        ],
    ))

fig.update_layout(
    updatemenus=[
        dict(
            type="dropdown",
            direction="down",
            buttons=buttons,
            x=0.0,
            y=1.15,
            xanchor="left",
            yanchor="top",
            showactive=True,
            bgcolor="#222222",
            bordercolor="#444444",
            font=dict(color="#f0f0f0", size=12)
        )
    ],
    margin=dict(t=110, r=20, l=60, b=40)
)

display(fig)
```

In [4]:

```
# Report count of tasks by sailing_id, role, and start_hour using prepare_ta
def get_report(ship_id, role_name):
    target_ship = ship_id
    target_role = role_name

    prepared, start_hours, _, y_vals, t0, s_id, r_name = prepare_tasks_for(t

    report_rows = [
        {"sailing_id": task["sailing_id"], "role": r_name, "start_hour": tas
         "count": count}
        for task, count in zip(prepared, y_vals)
    ]

    report_df = pd.DataFrame(report_rows)
    return report_df
```

```
data = []
for sh in ships:
    for rl in roles:
        data.append((sh, roles, max(get_report(sh, rl)["count"])))
pd.DataFrame(data, columns=["ship_id", "role", "max_count"])
```

Out[4]:

In [5]:
```
def get_tasks_by_sailing(ship_id, sailing_id):
    return {role: [t for t in tks if t["sailing_id"] == sailing_id] for role


selected_tasks = get_tasks_by_sailing('S1', 'S1-1')
```

In [6]:
```
def get_count(list_of_tasks):
    counts = {}
    for tup in list_of_tasks:
        counts.setdefault(tup, 0)
        counts[tup] += 1
    return counts


selected_3days_with_duplicates = {role: [(x['start_ts'], x['end_ts']) for x
                                         x['end_ts'] < pd.to_datetime('2025-
                                  selected_tasks}
selected_3days = {role: sorted(set(selected_3days_with_duplicates[role]), ke
                  selected_3days_with_duplicates}
counts = {role: get_count(selected_3days_with_duplicates[role]) for role in
required_personal = {role: [counts[role][x] for x in selected_3days[role]] f
```

In [7]:
```
import numpy as np


def gen_columns(rl, max_columns=None):
    tasks_sel = selected_3days[rl]
    number_of_tasks = len(tasks_sel)
    if number_of_tasks == 0:
        return np.zeros((0, 0), dtype=int)

    cols = []
    col = np.zeros(number_of_tasks, dtype=int)

    def dfs(i, next_one_allowed_at, has_one):
        if max_columns is not None and len(cols) >= max_columns:
```

```
            return
        if i == number_of_tasks:
            if has_one:
                cols.append(col.copy())
            return
        # place 0
        col[i] = 0
        dfs(i + 1, next_one_allowed_at, has_one)
        # place 1 if allowed (keep at least two zeros between ones)
        if i >= next_one_allowed_at:
            col[i] = 1
            dfs(i + 1, i + 3, True)
            col[i] = 0  # reset

    dfs(0, 0, False)
    if not cols:
        return np.zeros((number_of_tasks, 0), dtype=int)
    return np.stack(cols, axis=1)
```

In [8]:

```
import gurobipy as gp


class Solution:
    def __init__(self, role):
        self.role = role
        self.a_matrix = gen_columns(self.role)
        self.required_personal = np.array(required_personal[self.role])
        self.ordered_unique_tasks = selected_3days[self.role]
        self.model = gp.Model(self.role)
        self.variables = self.model.addMVar(self.a_matrix.shape[1], vtype=gp
        self.solution = None

    def solve(self):
        self.model.setObjective(self.variables.sum(), gp.GRB.MINIMIZE)
        self.model.addConstr(self.a_matrix @ self.variables == self.required
        self.model.optimize()
        self.model.write(f'{self.role}-sp.mps')
        self.model.write(f'{self.role}-sp.lp')
        return self

    def get_solution(self):
        selected_patterns = []
        for var in self.model.getVars():
            var_name = var.varName
            var_value = var.x
            if var_value > 0.5:
                selected_patterns.append(int(var_name.split("[")[1].split("]
                print(selected_patterns)
        self.solution = {"patterns": self.a_matrix[:, selected_patterns]}
        schedules = {
            f"schedule_{selected_patterns.index(i)}": [self.ordered_unique_t
            for i in selected_patterns
        }
        self.solution["schedules"] = schedules
        return self
```

```
    def run(self):
        self.solve()
        self.get_solution()
        return self
```

In [9]:
```
solutions = {role: Solution(role).run().solution for role in roles}
```

```
Restricted license - for non-production use only - expires 2026-11-23
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 24.04.3
LTS")
CPU model: 12th Gen Intel(R) Core(TM) i5-12500H, instruction set [SSE2|AVX|A
VX2]
Thread count: 16 physical cores, 16 logical processors, using up to 16 threa
ds
Optimize a model with 11 rows, 87 columns and 208 nonzeros
Model fingerprint: 0x4d9dd83c
Variable types: 0 continuous, 87 integer (87 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 11.0000000
Presolve time: 0.00s
Presolved: 11 rows, 87 columns, 208 nonzeros
Variable types: 0 continuous, 87 integer (87 binary)
Root relaxation: objective 3.000000e+00, 23 iterations, 0.00 seconds (0.00 w
ork units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
*    0     0               0       3.0000000    3.00000  0.00%     -    0s
Explored 1 nodes (23 simplex iterations) in 0.03 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)
Solution count 2: 3 11
Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.000
0%
[39]
[39, 58]
[39, 58, 86]
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 24.04.3
LTS")
CPU model: 12th Gen Intel(R) Core(TM) i5-12500H, instruction set [SSE2|AVX|A
VX2]
Thread count: 16 physical cores, 16 logical processors, using up to 16 threa
ds
Optimize a model with 11 rows, 87 columns and 208 nonzeros
Model fingerprint: 0x4d9dd83c
Variable types: 0 continuous, 87 integer (87 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 11.0000000
Presolve time: 0.00s
```

```
Presolved: 11 rows, 87 columns, 208 nonzeros
Variable types: 0 continuous, 87 integer (87 binary)
Root relaxation: objective 3.000000e+00, 23 iterations, 0.00 seconds (0.00 w
ork units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
*    0     0               0       3.0000000    3.00000  0.00%     -    0s
Explored 1 nodes (23 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)
Solution count 2: 3 11
Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.000
0%
[39]
[39, 58]
[39, 58, 86]
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 24.04.3
LTS")
CPU model: 12th Gen Intel(R) Core(TM) i5-12500H, instruction set [SSE2|AVX|A
VX2]
Thread count: 16 physical cores, 16 logical processors, using up to 16 threa
ds
Optimize a model with 11 rows, 87 columns and 208 nonzeros
Model fingerprint: 0x4d9dd83c
Variable types: 0 continuous, 87 integer (87 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 11.0000000
Presolve time: 0.00s
Presolved: 11 rows, 87 columns, 208 nonzeros
Variable types: 0 continuous, 87 integer (87 binary)
Root relaxation: objective 3.000000e+00, 23 iterations, 0.00 seconds (0.00 w
ork units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
*    0     0               0       3.0000000    3.00000  0.00%     -    0s
Explored 1 nodes (23 simplex iterations) in 0.03 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)
Solution count 2: 3 11
Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.000
0%
[39]
[39, 58]
[39, 58, 86]
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 24.04.3
LTS")
CPU model: 12th Gen Intel(R) Core(TM) i5-12500H, instruction set [SSE2|AVX|A
VX2]
Thread count: 16 physical cores, 16 logical processors, using up to 16 threa
ds
Optimize a model with 11 rows, 87 columns and 208 nonzeros
Model fingerprint: 0x4d9dd83c
```

```
Variable types: 0 continuous, 87 integer (87 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 11.0000000
Presolve time: 0.00s
Presolved: 11 rows, 87 columns, 208 nonzeros
Variable types: 0 continuous, 87 integer (87 binary)
Root relaxation: objective 3.000000e+00, 23 iterations, 0.00 seconds (0.00 w
ork units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
*    0     0               0       3.0000000    3.00000  0.00%     -    0s
Explored 1 nodes (23 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)
Solution count 2: 3 11
Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.000
0%
[39]
[39, 58]
[39, 58, 86]
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 24.04.3
LTS")
CPU model: 12th Gen Intel(R) Core(TM) i5-12500H, instruction set [SSE2|AVX|A
VX2]
Thread count: 16 physical cores, 16 logical processors, using up to 16 threa
ds
Optimize a model with 11 rows, 87 columns and 208 nonzeros
Model fingerprint: 0x4d9dd83c
Variable types: 0 continuous, 87 integer (87 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 11.0000000
Presolve time: 0.00s
Presolved: 11 rows, 87 columns, 208 nonzeros
Variable types: 0 continuous, 87 integer (87 binary)
Root relaxation: objective 3.000000e+00, 23 iterations, 0.00 seconds (0.00 w
ork units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
*    0     0               0       3.0000000    3.00000  0.00%     -    0s
Explored 1 nodes (23 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 16 (of 16 available processors)
Solution count 2: 3 11
Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.000
0%
[39]
[39, 58]
[39, 58, 86]
```

In [10]:
```python
import pandas as pd
import plotly.express as px

# --- Flatten your solutions dict into a DataFrame ---
records = []
for role, data in solutions.items():
    for schedule_name, intervals in data["schedules"].items():
        for start, end in intervals:
            records.append({
                "role": role,
                "schedule": schedule_name,
                "start": start,
                "end": end
            })

df = pd.DataFrame(records)
df = df.sort_values(by=["role", "schedule", "start"])

# --- Convert datetimes to hours since first start (relative time axis) ---
t0 = df["start"].min()
df["start_hours"] = (df["start"] - t0).dt.total_seconds() / 3600
df["duration_hours"] = (df["end"] - df["start"]).dt.total_seconds() / 3600

# --- Generate one plot per role ---
figs = {}
for role in df["role"].unique():
    role_df = df[df["role"] == role]
    fig = px.bar(
        role_df,
        x="duration_hours",
        y="schedule",
        base="start_hours",  # start of bar
        color="schedule",
        orientation="h",
        title=f"Schedules for Role: {role} (Hours since first start)"
    )
    fig.update_layout(
        xaxis_title="Time (hours)",
        yaxis_title="Schedules",
        bargap=0.3,
        height=500
    )
    fig.update_yaxes(autorange="reversed")  # keep first schedule on top
    figs[role] = fig

# --- Show one of them (for example, first role) ---
for rl in list(figs.keys()):
    display(figs[rl])
```

In [ ]:
```
```